POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Progetto ed implementazione di una funzione virtuale di sicurezza in rete per il filtraggio dei pacchetti in ambiente NFV

Relatori
prof. Antonio Lioy
dott. Marco De Benedictis

Candidato
Antonio Checola

Dicembre 2019

Sommario

Le moderne reti di telecomunicazione presentano un gran numero di dispositivi hardware di proprietà utilizzati per fornire agli utenti servizi di vario tipo in base alle esigenze. La connettività è ormai diventata un requisito che deve essere garantito ad un numero di dispositivi sempre più ampio e sempre più eterogeneo. Questo aspetto ha impatto sul campo della sicurezza informatica. Con l'avanzare del tempo infatti, si ha un numero crescente di dispositivi connessi che fornisce una superficie di attacco sempre piú vasta. Inoltre i progressi apportati in campo tecnologico mettono a disposizione dei criminali informatici risorse di attacco sempre più all'avanguardia. Ciò comporta la richiesta di un continuo adattamento dei servizi di sicurezza offerti. Esso risulta spesso lento e costoso a causa della necessità di installare nuove apparecchiature hardware, oltre a comportare difficoltà nella ricerca di spazio disponibile per collocare nuovi dispositivi.

In tale contesto, viene introdotto il paradigma Network Function Virtualisation (NFV) , il quale introduce un concetto di rete orientato al software tramite l'utilizzo della virtualizzazione delle funzioni. I blocchi fisici classici vengono sostituiti dalle Virtual Network Function (VNF) in grado di essere eseguite su hardware generico, riducendo l'utilizzo di dispositivi specifici alla singola funzione. Tale aspetto viene integrato con il concetto di Cloud Computing, il quale permette di allocare risorse su richiesta ed in modo distribuito. Ciò permette di ottenere una gestione della rete efficiente e con riutilizzo delle capacità a disposizione. Nel lavoro di tesi si è voluto contribuire a tale scenario implementando una funzione virtuale che sia in grado di eseguire il filtraggio dei pacchetti in ambiente NFV. Come tecnologia per la realizzazione del packet filter è stato utilizzato il software open-source "PF". Per essa sono stati progettati e sviluppati moduli che permettano l'inizializzazione della funzione realizzata in modo da essere utilizzata in ambito NFV.

Tra le problematiche relative alle funzioni di sicurezza rientra quello relativo alla configurazione di tali servizi. A tal proposito durante la tesi è stato introdotto un approccio basato su politiche di sicurezza atto a definire una metodologia per facilitare la configurazione delle funzioni utilizzate. Nello studio svolto si sono individuati dei linguaggi di astrazione delle politiche che partono da definizioni di alto livello fino ad arrivare alla configurazione a basso livello. Tale approccio ben si integra nel mondo NFV, in quanto riesce a definire le politiche con un certo livello di astrazione indipendente dalla tecnologia utilizzata, permettendo una configurazione agnostica dall'implementazione scelta. Per la gestione delle regole del packet filter viene adottato tale approccio. A tal proposito viene implementato all'interno della funzione di rete un modulo che sia in grado di ricevere le richieste nel linguaggio di astrazione analizzato e di trasformarle in configurazioni di basso livello per la tecnologia PF. Per verificare la funzione di rete virtuale realizzata viene introdotto un ambiente che sia in grado di simulare il contesto NFV, tramite il quale si procede con la validazione della VNF realizzata e con l'esecuzione di test per la valutazione delle performance raggiunte.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Oggigiorno le reti internet presentano un vasto numero di dispositivi hardware di proprietà, utilizzati per fornire i più vari servizi alle diverse tipologie di utente a seconda delle loro necessità. Con l'avanzare dei tempi la connettività viene richiesta in qualunque luogo ed in ogni momento, portando ad un incremento del numero di dispositivi connessi. Un maggior numero di dispositivi connessi richiede un maggior numero di risorse per la gestione delle funzionalità necessarie. Per gli *Internet Service Provider* (ISP) diventa sempre più complicato trovare nuovi spazi e nuove risorse energetiche per poter installare hardware in grado di soddisfare tali esigenze. In un contesto del genere è risultato necessario un nuovo approccio al design delle reti di telecomunicazione, che ha portato all'introduzione dei concetti di *Software Defined Networking* (SDN), *Network Function Virtualisation* (NFV) e *Cloud Computing*.

Il *Cloud Computing* rappresenta la tecnologia abilitante alla virtualizzazione della rete fornendo un nuovo modello di fruizione dei servizi. Attraverso la rete il fornitore configura su richiesta risorse pre-esistenti distribuite geograficamente, le quali saranno disponibili in remoto ai consumatori del servizio. SDN fornisce un nuovo approccio all'architettura di rete. L'obiettivo è ottenere una rete programmabile tramite il disaccoppiamento del *control plane* di una funzione di rete ed il relativo *data plane*. A tal proposito la logica di controllo viene spostata al di fuori dei dispositivi che compongono la rete, sfruttando un componente esterno che prende il nome di *SDN Controller*. NFV è invece un paradigma che prevede la distribuzione delle funzionalità di rete sotto forma di blocchi software, denominati *Virtual Network Functions (VNFs)*. Con l'utilizzo di una VNF è possibile eseguire i servizi forniti su macchine virtuali ospitate su nodi generici, evitando l'installazione di hardware specifico ad una singola funzionalità. Alcuni classici esempi applicativi prevedono la realizzazione di *load balancer*, *firewall* e *router* virtuali. L'architettura NFV, insieme al concetto di SDN, porta ad un disaccoppiamento tra i servizi offerti dai nodi e gli apparati hardware su cui essi risiedono. I classici blocchi funzionali fisici della rete, costituiti dai *Physical Network Functions* (PNFs), vengono sostituiti da nodi generici su cui è possibile eseguire le VNF. È inoltre possibile combinare le VNF in modo da fornire catene di servizi complesse, denominate *Service Function Chains* (SFC). Questa evoluzione nel design permette una gestione flessibile e programmabile dei servizi forniti ed un riutilizzo efficiente delle risorse a disposizione, rendendo possibile l'inserimento, la rimozione e lo spostamento delle funzionalità all'interno dell'infrastruttura in base alle esigenze degli operatori o degli utenti senza dover installare nuovi apparati hardware e con tempistiche brevi. Inoltre, tale gestione introduce una riduzione nei costi complessivi degli ISP, sia per quanto riguarda i costi per la fornitura, anche definiti *CAPital EXpenditure* (CAPEX), sia per i costi di mantenimento di un servizio di rete, definiti con il termine *OPerating EXpenditure* (OPEX).

L'architettura NFV introduce risvolti importanti anche nel campo della sicurezza informatica. Il tema della *cyber security* è più attuale che mai ed è causa di ingenti esborsi economici, dovuti sia ad investimenti per la fornitura di protezioni adeguate sia a perdite dovute ad attacchi informatici. Secondo i dati raccolti nel report del *Federal Bureau of Investigation* (FBI) del 2018 [1], mostrati in Fig. 1.1, il numero totale di denunce relative a crimini informatici è aumentato, così come le perdite economiche, che hanno raggiunto un totale di $7.45 bilioni dal 2014 al 2018. Da questi numeri risulta evidente che in un mondo in cui il numero di dispositivi connessi è in

continuo aumento, la superficie di attacco dei criminali informatici risulta sempre più vasta. In questo scenario le tipologie di attacco continuano ad evolversi rapidamente. I criminali hanno a disposizione molti più *target* su cui effettuare attacchi e su cui sperimentare nuove tecniche, oltre ad avere a disposizione una potenza computazionale sempre maggiore grazie ai progressi apportati in ambito tecnologico. Proprio a causa di questa continua evoluzione, è richiesto un

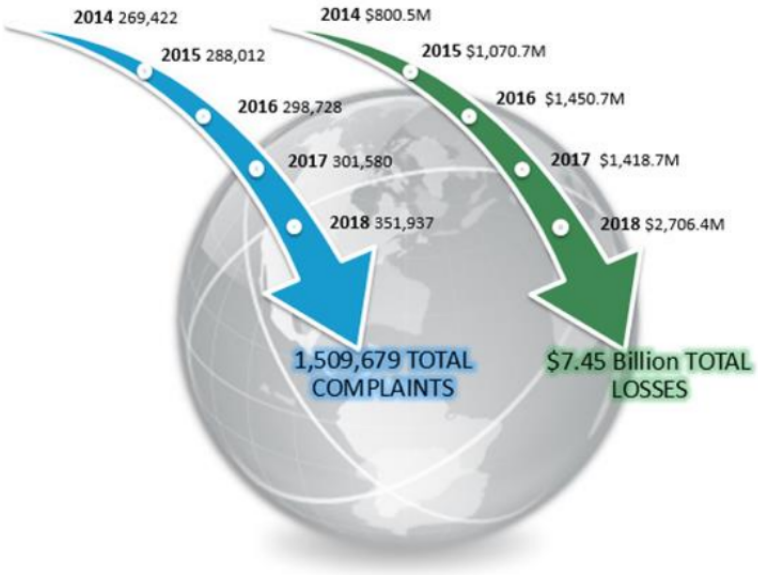

Figura 1.1. Dati sui crimini informatici raccolti nel report FBI 2018

adattamento altrettanto rapido nelle funzionalità di sicurezza fornite agli utenti. L'approccio classico che prevede l'installazione di nuove risorse ad-hoc risulta costoso, poco flessibile e con tempistiche di reazione elevate. A tal proposito, NFV mette a disposizione degli ISP il paradigma *Security-as-a-Service* (SecaaS). In questo scenario gli ISP possono sfruttare il concetto di *Virtual Network Security Functions* (vNSFs). Le vNSF forniscono una specializzazione *security-oriented* delle VNF, in grado di agire sulla rete in modo attivo (funzioni per contrastare possibili attacchi) o passivo (funzioni di monitoraggio e raccolta dati). L'utilizzo delle vNSF applica al campo della sicurezza informatica i vantaggi descritti per il paradigma NFV, fornendo una soluzione in grado di adattarsi rapidamente ai continui cambiamenti della criminalità informatica, con una notevole riduzione dei costi nello sviluppo di nuovi servizi.

Questo lavoro di tesi pone la sua attenzione sulla realizzazione del prototipo di una vNSF che realizzi le funzionalità di *Packet Filter stateful*. Tale vNSF deve essere in grado di effettuare il filtraggio del traffico in base alle informazioni contenute nell'intestazione dei pacchetti elaborati. In particolare si sfruttano le informazioni appartenenti al livello 3 del modello *Open System Interconnection* (OSI), analizzando l'*Internet Protocol* (IP) header ed informazioni appartenenti al livello 4, come porta sorgente/destinazione e protocollo di trasporto. A tale scopo si andranno ad identificare e sviluppare i blocchi necessari alla gestione e configurazione del *Packet Filter* nel contesto NFV. Verranno inoltre studiate ed utilizzate tecnologie in grado di simulare una architettura NFV per verificare il funzionamento della vNSF in tale contesto applicativo.

Nel lavoro di tesi sono presenti i seguenti capitoli:

- **Capitolo 2:** introduce i concetti essenziali presenti nel contesto della tesi, per poter avere una migliore comprensione del lavoro realizzato.
- **Capitolo 3:** fornisce una visione dettagliata delle tecnologie utilizzate per realizzare lo scenario in cui viene eseguita la vNSF ed il modo in cui esse interagiscono.
- **Capitolo 4:** analizza i moduli realizzati per garantire il funzionamento della vNSF da un punto di vista implementativo. Viene inoltre presentato un test funzionale atto a mostrare il corretto funzionamento della vNSF, ed un test di analisi delle prestazioni.

- **Capitolo 5:** conclusioni maturate con il lavoro di tesi.
- **Appendice A:** manuale utente.
- **Appendice B:** manuale dello sviluppatore.

Capitolo 2

Background

Questo capitolo analizza i concetti introduttivi essenziali alla realizzazione della vNSF, come NFV, Cloud Computing e SDN, fondamentali per la virtualizzazione delle funzionalità presenti sulla rete, ed il concetto di *Packet Filter*, per fornire una panoramica più dettagliata delle funzionalità che la vNSF realizzata implementa. Viene inoltre fornita una panoramica sul metodo di configurazione basato su policy di sicurezza, utilizzato per le funzioni di rete.

2.1 Network Function Virtualisation

Con NFV [2] si intende il paradigma che utilizza tecnologie di virtualizzazione per fornire funzionalità di rete, in modo da ottenere nuovi metodi di design, di sviluppo e di gestione dei servizi, abbandonando i classici dispositivi hardware. Nell'architettura NFV, come mostrato in Fig. 2.1, il servizio viene scomposto in uno o più blocchi logici, realizzati dalle VNF, ossia moduli software in grado di essere eseguiti su hardware *general purpose*, senza l'installazione di dispositivi specializzati nelle singole funzionalità. Tale gestione permette una modellazione dinamica della rete, in quanto le VNF possono essere spostate, a seconda dei bisogni di utenti e operatori, tra i nodi che costituiscono l'infrastruttura.

Con NFV si intende dunque eliminare la dipendenza tra funzionalità di rete e hardware specifico, ottenendo come vantaggi [3]:

- Ottimizzazione dell'uso delle risorse, potendo utilizzare lo stesso server fisico per più funzioni di rete, in modo da poter sfruttare al meglio le capacità della rete.
- Gestione della capacità allocata in base al carico effettivo della rete, incrementando o riducendo il numero di risorse utilizzate.
- Garantire l'affidabilità, potendo riallocare le funzioni richieste in caso di malfunzionamenti dell'hardware.
- Riconfigurare facilmente e rapidamente la topologia della rete per ottimizzarne le performance.

Oltre ai vantaggi indicati, si possono evidenziare alcune criticità, tra cui:

- Necessità di realizzare un meccanismo di gestione della VNF all'interno dell'architettura di rete, in quanto la VNF non può essere realizzata semplicemente effettuando un *porting* su ambiente virtuale della funzione che si desidera realizzare.
- Necessità per gli operatori di avere una piattaforma di orchestrazione che sia in grado di gestire le VNF.

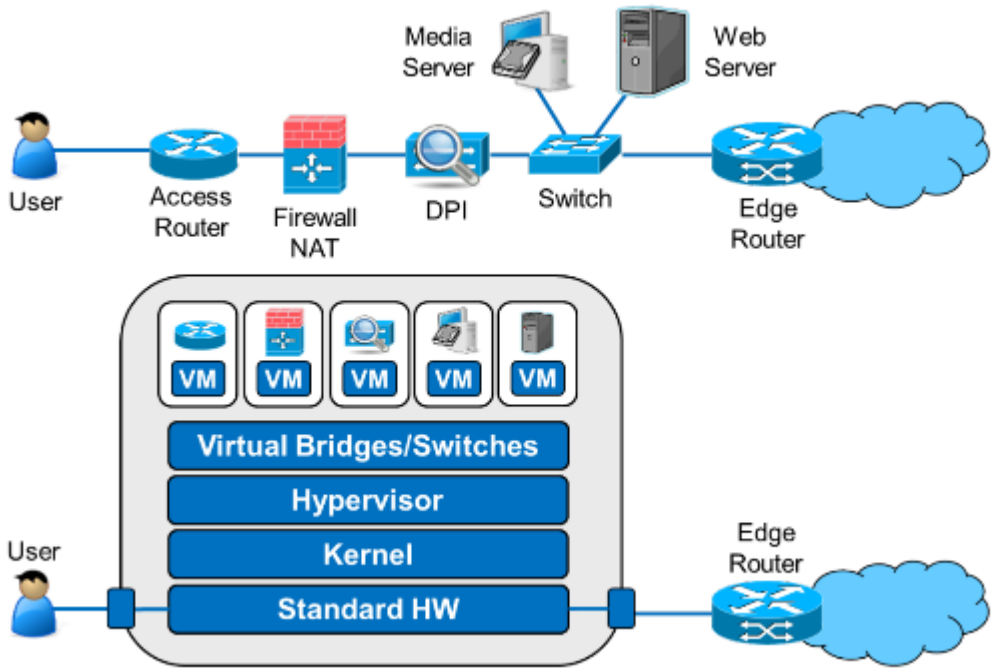

Figura 2.1. Confronto rete con NFV e con hardware specifico

2.1.1 L'architettura NFV

In questo paragrafo verrà illustrata l'architettura descritta dall'*European Telecommunications Standards Institute* (ETSI) per la realizzazione della tecnologia NFV. È possibile partire da una visione ad alto livello dell'architettura, mostrata in Fig. 2.2, in cui si vanno ad identificare tre moduli principali all'interno della tecnologia NFV, identificati dalle VNF, la *NFV Infrastructure* (NFVI) e il modulo *NFV Management And Orchestration* (NFV MANO). Le VNF forniscono le implementazioni del software che può essere eseguito all'interno dell'infrastruttura, rispettando requisiti approfonditi nella sezione 2.1.2. NFVI rappresenta l'insieme delle risorse a disposizione delle VNF che rendono possibile l'esecuzione di tali funzioni virtualizzate. L'infrastruttura fornisce le risorse hardware astratte tramite uno strato di virtualizzazione, in modo da essere utilizzabili dalle VNF ospitate sull'infrastruttura. NFV MANO rappresenta il modulo atto all'orchestrazione delle risorse fisiche e software, ed alla gestione del loro ciclo di vita. L'architettura studiata per NFV permette di avere un controllo dinamico delle VNF istanziate sulla rete e di definire le relazioni tra esse, fornendo i servizi richiesti in modo rapido e flessibile.

I moduli principali possono essere suddivisi in alcuni blocchi fondamentali, fornendo una visione più dettagliata dei componenti che permettono la realizzazione della tecnologia NFV, mostrata in Fig. 2.3. I blocchi funzionali introdotti con questa nuova prospettiva sono:

- **Element Management System (EMS):** effettua le funzionalità di gestione delle VNF.
- **Virtualised Infrastructure Manager (VIM):** comprende le funzionalità che vengono usate per controllare e gestire l'interazione di una VNF con le risorse di calcolo, di storage e di rete, oltre a gestire la virtualizzazione della VNF.
- **Orchestratore:** gestisce l'infrastruttura NFV e le risorse software.
- **VNF Manager:** gestisce il ciclo di vita delle VNF.
- **Data Model:** rappresenta l'insieme dei dati che permettono di caratterizzare gli elementi utilizzati dall'architettura NFV, comprendendo il modo in cui le VNF debbano essere attraversate per fornire un *Network Sevice* (NS).

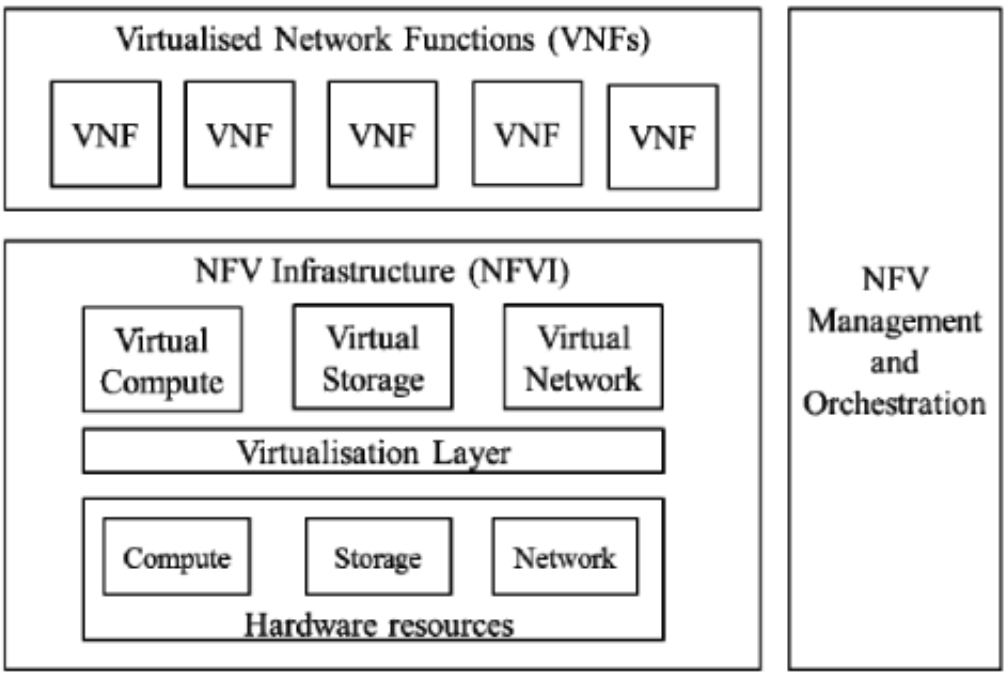

Figura 2.2. Framework NFV. Visione ad alto livello

- **OSS/BSS:** sistema utilizzato dall'operatore per supportare i servizi di telecomunicazione.

2.1.2 Virtual Network Function

Una VNF rappresenta una delle unità principali per la realizzazione dell'architettura descritta nel paragrafo 2.1.1. Essa fornisce l'implementazione software di un funzione di rete, eseguibile sull'infrastruttura NFV e gestita durante il suo ciclo di vita da un orchestratore ed un gestore di VNF, ossia dal *NFV Orchestrator* (NFVO) e dal *NFV Manager* (NFVM). Le VNF specializzate in funzioni relative alla sicurezza della rete (es. firewall, IDS, etc..) prendono il nome di vNSF, le quali si distinguono in passive o attive. Le vNSF passive vengono utilizzate per monitorare la rete e per raccogliere dati sull'infrastruttura, fornendo informazioni utili agli ISP. Le vNSF attive vengono utilizzate in risposta ad una minaccia informatica, in modo da contrastare il rischio introdotto. Le VNF, allo scopo di interagire con NFVO e NFVM, seguono un'architettura definita, con interfacce che permettono l'interazione interna alla VNF e con altri componenti atti al funzionamento del paradigma NFV, tale architettura è mostrata nella Fig. 2.5.

Una VNF, durante la fase di progettazione, si trova a dover rispettare determinati requisiti in termini di performance, scalabilità e funzionalità da realizzare, che la portano ad essere organizzata in uno o più moduli, che prendono il nome di *Virtual Network Function Component* (VNFC). Una singola VNF può essere implementata in modo differente da diversi fornitori. I VNFC che compongono la singola funzione virtualizzata sono connessi in un grafo, come mostrato in Fig. 2.4. Inoltre i singoli componenti sono progettati in modo da essere parallelizzabili, dando quindi la possibilità di essere istanziati più volte per una singola istanza della VNF, o non parallelizzabili. Facendo riferimento alla Fig. 2.5, una VNF presenta diversi tipi di interfaccia, divisibili in base alla loro funzionalità nei seguenti gruppi:

- **SWA-1:** fornisce un'interfaccia che abilita la comunicazione tra le VNF, sia all'interno dello stesso servizio di rete sia tra servizi differenti, in modo da ottenere funzionalità di rete più complesse. Una VNF può supportare interfacce multiple di questo tipo.

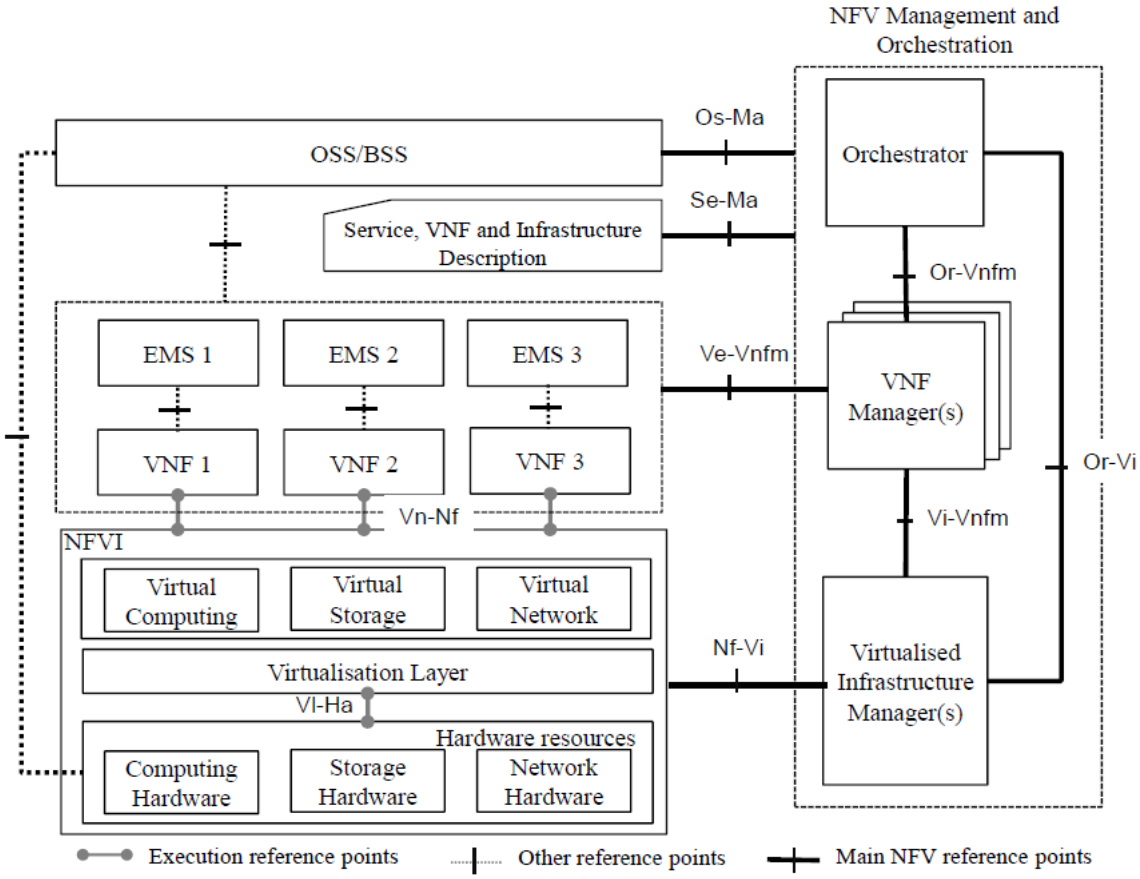

Figura 2.3. Framework architetturale NFV di riferimento

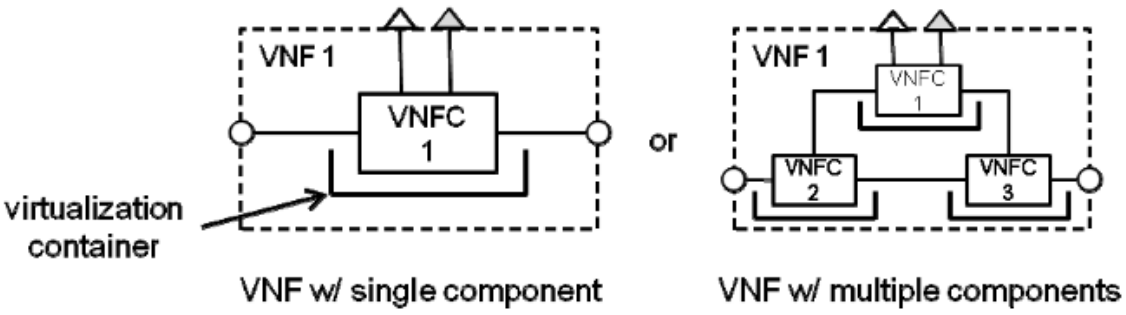

Figura 2.4. Esempi di composizione di una VNF

- **SWA-2:** fornisce l'insieme delle interfacce interne alla VNF, utilizzate dai VNFC che compongono la funzione virtualizzata per comunicare tra loro. Il requisito più importante per le interfacce appartenenti a questo insieme è rappresentato dalle performance. Esse sono solitamente trasparenti agli utilizzatori della VNF.
- **SWA-3:** fornisce l'interfaccia con la parte di gestione della VNF, permettendo lo scambio di messaggi con l'orchestratore e con il VNF manager.
- **SWA-4:** fornisce l'interfaccia utilizzata dall'EM per effettuare una gestione a *runtime* della VNF.

- **SWA-5:** fornisce l'insieme delle interfacce che permettono la comunicazione di una VNF con l'infrastruttura su cui la funzione viene virtualizzata (NFVI).

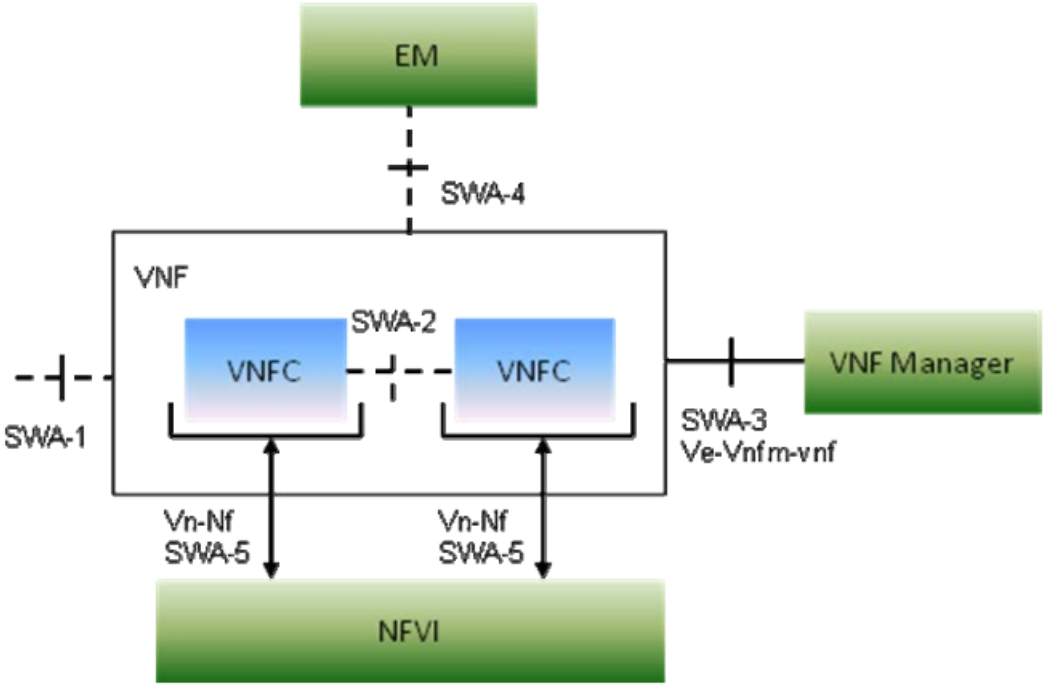

Figura 2.5. Interfacce di una VNF

2.1.3 Security as a Service

Il caso d'uso di interesse nel contesto del lavoro di tesi è quello che l'ETSI identifica come "Security as a Service" (SecaaS) [4]. Questo caso d'uso nasce in risposta alla notevole crescita della criminalità informatica. Le tecniche di attacco si sono evolute in termini di complessità e potenza, ed è probabile che la situazione peggiorerà in futuro per motivi come:

- **Banda larga:** Con l'utilizzo della fibra ottica e l'introduzione delle reti 5G vengono fornite capacità di banda sempre più elevate ai dispositivi connessi. È ormai comune la fornitura di accessi a 100 Mbit/s per utenti residenziali. Per i criminali informatici è possibile sfruttare questi punti di accesso con alta capacità di banda per mandare dati e saturare le risorse di un sistema informatico, portando attacchi di tipo *Distributed Denial of Service* (DDoS).
- **Evoluzione dell'Internet of Things (IoT):** il numero di dispositivi IoT connessi continua a crescere in numero. Questi dispositivi si distinguono per eterogeneità, basso costo e carenza di aggiornamenti. Proprio a causa di aggiornamenti poco frequenti, questi dispositivi presentano spesso delle vulnerabilità sfruttabili tramite attacchi informatici per accedere a risorse e dati sensibili.
- **Aziende dipendenti da Internet:** la connettività rappresenta ormai un requisito aziendale indispensabile. Inoltre alcune aziende nascono proprio con lo scopo di fornire servizi tramite la rete. Le risorse necessarie alla protezione dai rischi informatici può eccedere le capacità a disposizione, rendendo l'azienda stessa una vittima perfetta dei criminali informatici.

Come conseguenza è richiesto un continuo adattamento delle difese informatiche utilizzate dalle organizzazioni. Il processo di aggiornamento risulta spesso troppo lento rispetto alla velocità di cambiamento delle minacce informatiche, oltre a risultare estremamente costoso. Il caso d'uso in

analisi spiega come la tecnologia NFV possa fornire una soluzione che sia in grado di adattarsi al continuo evolversi di questo ambiente, fornendo funzioni virtualizzate specializzate nel contesto della sicurezza informatica, identificate dalle vNSF. L'utilizzo di una soluzione basata su NFV riesce a fornire:

- Risposta ad-hoc e veloce ad ogni tipo di minaccia informatica sulla rete attraverso l'utilizzo di specifiche vNSF.
- Possibilità di scalare le risorse oltre le disponibilità della singola azienda.
- Analisi della rete con operazioni di monitoraggio dei dati di sicurezza.

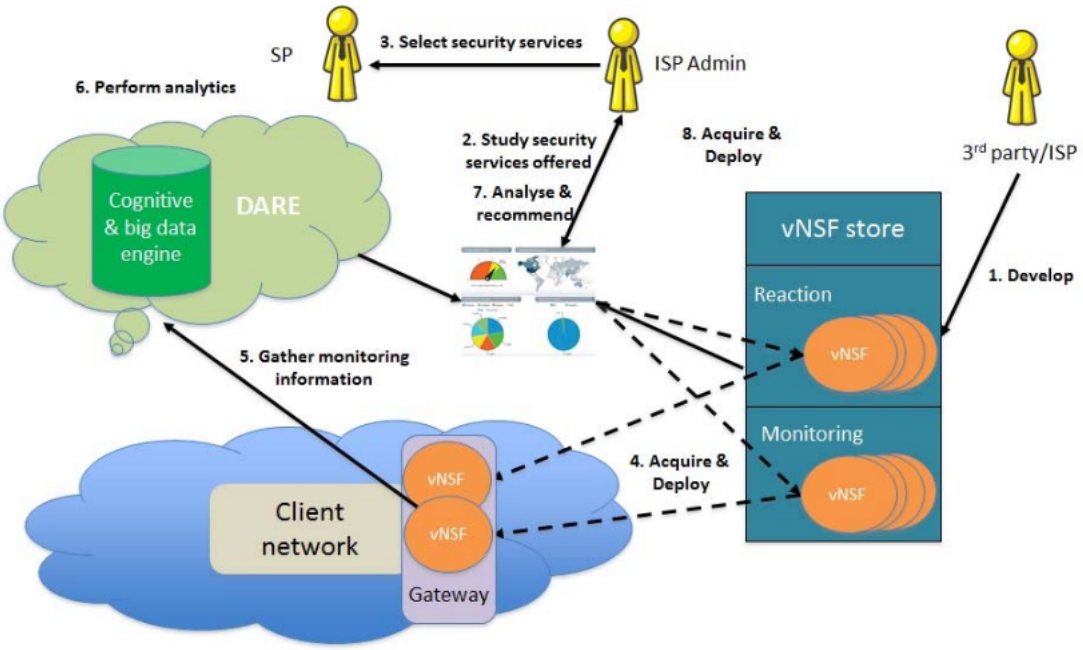

Figura 2.6. Architettura SecaaS

In questo caso d'uso vengono identificati tre protagonisti principali:

- **Clienti:** identificano la parte di utilizzatori della rete che richiede un determinato servizio di sicurezza.
- **ISP:** forniscono il servizio tramite l'utilizzo della NFVI.
- **Sviluppatori:** forniscono l'implementazione della vNSF che viene pubblicata sul *vNSF store* e messa a disposizione degli utenti.

In tale contesto l'ISP ottiene una gestione centralizzata delle funzionalità di sicurezza, astraendo al cliente la complessità dell'analisi di sicurezza. All'utilizzatore della rete viene lasciata solo la scelta dei servizi di sicurezza di cui usufruire e le vNSF vengono inserite o rimosse dall'ISP direttamente all'interno dell'infrastruttura di rete o del gateway fornito. Inoltre il fornitore del servizio è l'unico responsabile della gestione e dell'aggiornamento della funzionalità. Questo approccio permette all'ISP di rispondere in modo rapido e flessibile alle esigenze del cliente tramite l'orchestrazione NFV e l'utilizzo di vNSF. Inoltre, in questo scenario centralizzato, l'ISP diventa anche il nucleo di raccolta ed elaborazione dei dati raccolti dalle vNSF. Ciò permette al fornitore di avere una migliore visione dell'infrastruttura e di effettuare migliori analisi e predizioni dei rischi.

Per la realizzazione del caso d'uso descritto vengono identificati i seguenti componenti principali:

- **vNSF**
- **Data Analysis and Remediation Engine (DARE):** componente che effettua monitoraggio delle minacce e utilizza tecniche di intelligenza cognitiva sfruttando l'analisi dei dati ricevuti dalle vNSF. Il DARE non deve essere necessariamente costituito da un unico componente, ma può anche essere distribuito su più elementi funzionali.
- **vNSF store:** catalogo centralizzato che permette di scegliere e selezionare le vNSF messe a disposizione.
- **vNSF orchestrator:** funzionalità associata allo stack NFV MANO, in grado di gestire in modo efficiente il ciclo di vita delle vNSF ed orchestrare le politiche di sicurezza.
- **vNSF e infrastruttura di attestazione:** componente che associa le vNSF e la configurazione di rete all'orchestrazione della rete, in modo da certificare l'affidabilità del servizio.

Il flusso seguito dal caso d'uso SecaaS, mostrato in Fig. 2.6, comprende i seguenti punti:

1. **Sviluppo:** in questa fase le vNSF vengono sviluppate dagli operatori o da terzi. Quando la vNSF è completa ed è stata testata, viene resa disponibile sul vNSF store.
2. **Analisi dei servizi di sicurezza offerti:** il cliente analizza i servizi offerti dal fornitore basandosi sulle proprie esigenze.
3. **Selezione dei servizi di sicurezza:** il cliente sceglie i servizi di sicurezza dei quali desidera usufruire.
4. **Deploy:** i servizi scelti dal cliente vengono forniti dall'ISP. Il servizio può consistere in una o più vNSF.
5. **Raccolta delle informazioni di monitoraggio:** le vNSF inviano al DARE i dati raccolti tramite il monitoraggio dell'infrastruttura. Il DARE acquisisce e valida i dati prima di conservarli e processarli.
6. **Analisi dei dati:** il DARE effettua l'analisi dei dati raccolti, basandosi sulle esigenze dei servizi di sicurezza utilizzati dai clienti.
7. **Risultati dell'analisi e consigli:** le informazioni raccolte vengono fornite ai clienti. Oltre ai dati raccolti, vengono forniti dei consigli su eventuali azioni extra da effettuare, basati su un'analisi che incrocia i dati rilevati sulle minacce della rete e i requisiti di sicurezza del cliente.
8. **Deploy post-analisi:** in base ai consigli dati il cliente può scegliere di usufruire di più vNSF, le quali vengono fornite dall'ISP.

2.1.4 Cloud Computing e Software Defined Networking

Il paradigma NFV, al fine di raggiungere una gestione flessibile della rete ed un utilizzo ottimizzato delle risorse a disposizione, si lega con i concetti di Cloud Computing e SDN.

Cloud Computing

Il *National Institute of Standards and Technology* (NIST) definisce il Cloud Computing come un modello per abilitare un accesso diffuso, conveniente e on-demand ad un pool di risorse configurabili facilmente assegnabili e rilasciabili. Il tutto con il minimo sforzo di gestione e con la minima interazione con il fornitore del servizio [5]. Per il modello cloud si vanno a descrivere cinque caratteristiche essenziali, tre modelli di servizio e quattro modelli di distribuzione. Le caratteristiche descritte sono:

- **Self-service on-demand:** un cliente, quando richiesto, deve poter acquisire le risorse necessarie in modo unilaterale, senza interazioni umane.
- **Ampio accesso alla rete:** le risorse sono accessibili attraverso la rete e attraverso metodi standard che permettono una grande eterogeneità delle piattaforme client (e.g. tablet, laptop, smartphone).
- **Pooling di risorse:** le risorse vengono raggruppate in modo da riuscire a servire più clienti ed assegnate in modo dinamico in base alle richieste. Il cliente non è a conoscenza dell'effettiva locazione delle risorse che gli sono state assegnate, anche se in alcuni casi può essere possibile specificare ad un livello più alto (e.g. paese, stato) la locazione delle risorse stesse.
- **Elasticità:** le risorse possono essere fornite e rilasciate in modo flessibile ed in alcuni casi automatico. In questo modo si ottiene una gestione ottimale delle capacità fornite, basandosi sulla domanda effettiva.
- **Servizio misurato:** i sistemi cloud controllano e ottimizzano l'utilizzo delle risorse tramite l'uso di metriche adatte al tipo di servizio fornito.

I modelli di servizio, per i quali vengono forniti alcuni esempi in Fig. 2.7, sono:

- **Software as a Service (SaaS):** si fornisce all'utente la possibilità di utilizzare le applicazioni di un fornitore eseguite su una infrastruttura cloud.
- **Platform as a Service (PaaS):** si fornisce all'utente la possibilità di istanziare, attraverso l'infrastruttura cloud, applicazioni create dal consumatore stesso. Gli applicativi devono essere sviluppati utilizzando strumenti, linguaggi di programmazione o librerie messe a disposizione dal fornitore del servizio in modo da essere compatibili con l'infrastruttura su cui vengono eseguiti.
- **Infrastructure as a Service (IaaS):** si fornisce all'utente la possibilità di utilizzare risorse computazionali, di storage o di altro tipo per poter eseguire software arbitrario.

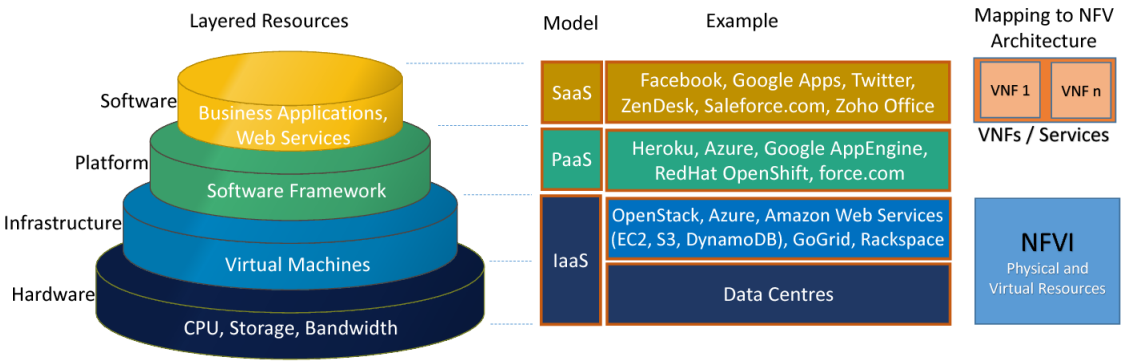

Figura 2.7. Modelli di servizio cloud

I modelli di distribuzione si distinguono in:

- **Cloud privato:** l'infrastruttura cloud ha un utilizzo esclusivo per una singola organizzazione.
- **Cloud comunitario:** l'infrastruttura cloud viene utilizzata in modo esclusivo da una comunità di utenti di diverse organizzazioni che condividono gli stessi interessi.
- **Cloud pubblico:** l'infrastruttura cloud è aperta all'utilizzo di qualsiasi utente.
- **Cloud ibrido:** l'infrastruttura cloud comprende al suo interno più tipologie di infrastrutture.

SDN

SDN [6] è un paradigma che fornisce una gestione della rete scalabile e dinamica. Questo obiettivo viene raggiunto tramite il disaccoppiamento tra *control plane* e *data plane*. Con control plane si intende la parte di sistema che effettua le decisioni su come effettuare il forwarding dei pacchetti. Il data plane è invece la parte del sistema che fisicamente riceve il pacchetto e provvede al suo smistamento. Nell'architettura SDN, mostrata in Fig. 2.8, si possono identificare tre livelli logici:

- **Application layer:** questo livello contiene servizi che gestiscono la rete ad un livello di astrazione alto. Queste applicazioni software comunicano con il livello di controllo per configurare l'infrastruttura sottostante.
- **Control layer:** questo livello contiene l'SDN Controller. Lo strato di controllo mantiene una visione globale della rete e comunica con gli altri strati tramite due interfacce. Attraverso la *Northbound Interface* (NBI) comunica con il livello applicativo e riceve i comandi da effettuare sul layer sottostante. Tramite la *Southbound Interface* (SBI) questi comandi vengono inviati al livello di infrastruttura per applicare nuove configurazioni all'interno dei dispositivi. La realizzazione della SBI è fornita da un insieme di istruzioni standardizzato per la gestione dell'hardware di rete. Un esempio è fornito dal protocollo OpenFlow [7], standardizzato dalla *Open Networking Foundation*. Al contrario per la NBI non esiste un set di comandi universale poichè la modalità e la frequenza con cui le applicazioni inviano richieste è variabile.
- **Infrastructure layer:** questo livello contiene gli switch, i router e gli altri strumenti di rete. Nel contesto SDN questi apparecchi sono programmabili dinamicamente.

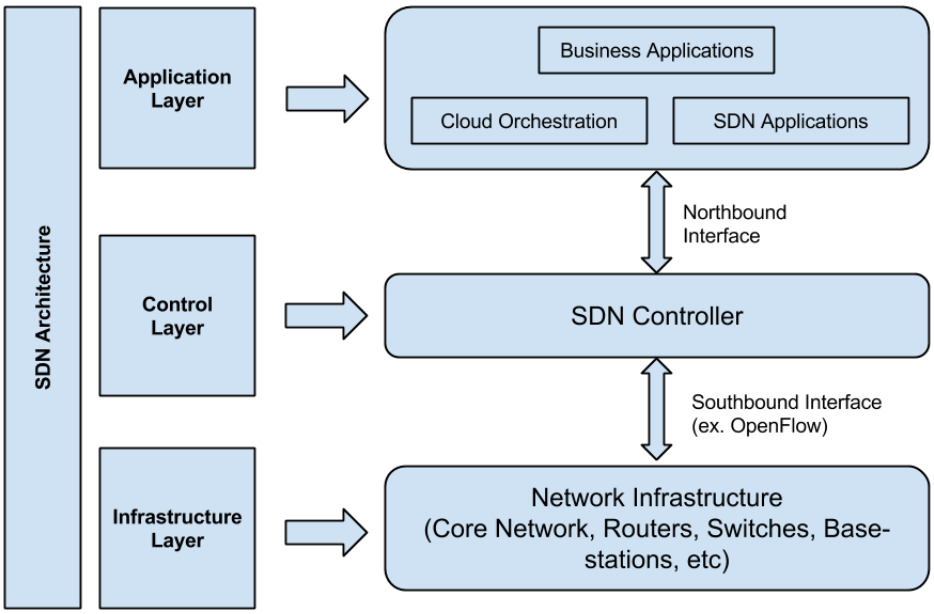

Figura 2.8. Livelli logici dell'architettura SDN

Oltre ai livelli logici, in una architettura SDN è possibile definire alcune caratteristiche fondamentali [8]:

- **Programmabilità:** costituisce l'aspetto fondamentale di SDN. Rappresenta il disaccoppiamento descritto in precedenza e fornisce la possibilità di programmare i dispositivi in base a specifici scenari di utilizzo.
- **Indipendenza dal protocollo:** permette a SDN di essere eseguito in modo congiunto con un vasto numero di tecnologie e protocolli di rete.

- **Modifica dinamica della rete:** questa caratteristica evidenzia la possibilità di SDN di modificare i parametri della rete in modo dinamico. Ciò deve essere possibile sia in caso di modifiche poco frequenti sia in caso di modifiche frequenti in cui è richiesto un tempo di risposta quasi real-time.
- **Granularità:** SDN deve essere in grado di gestire il traffico dei dati a vari livelli di granularità, dalla singola connessione fino alle connessioni di rete più complesse. In questo modo si garantisce scalabilità ed il control layer più agire su livelli differenti.
- **Elasticità:** SDN deve avere l'abilità di incrementare o decementare l'utilizzo delle risorse di rete in base all'effettiva capacità richiesta. Ciò permette al control plane di reagire in modo efficace alle varie situazioni di traffico.

Queste caratteristiche hanno permesso ad SDN di avere un notevole successo nel mondo delle telecomunicazioni, in quanto questo paradigma permette una gestione ottimale della rete.

Relazione tra NFV, SDN e Cloud Computing

Il paradigma NFV, come anticipato, è legato ai concetti di SDN e Cloud Computing con una relazione che può essere riassunta in Fig. 2.9. Il concetto che lega questi tre paradigmi è la virtualizzazione, intesa a diversi livelli. Nel cloud computing si effettua la virtualizzazione delle risorse computazionali, in SDN si virtualizza la gestione della rete, mentre in NFV si virtualizzano le funzioni di rete. Le funzioni sviluppate nel contesto NFV non si limitano ai soli servizi di telecomunicazione. D'altra parte, le VNF costituiscono solo un sottoinsieme del caso d'uso SaaS introdotto per il Cloud Computing, come mostrato in Fig. 2.7. Il cloud ricopre però un ruolo importante nello sviluppo di funzioni virtualizzate. Come visto per il caso d'uso IaaS, è possibile accedere a risorse virtualizzate messe a disposizione da fornitori attraverso la rete. Questo costituisce un ambiente di sviluppo facilmente scalabile, flessibile e poco costoso che risulta ideale per lo sviluppo di nuove funzioni virtualizzate. Ciò permette di implementare e sperimentare velocemente nuove VNF efficienti, contribuendo a spostare il mondo delle telecomunicazioni verso un ambiente virtualizzato. Anche i concetti di NFV e SDN possono essere profondamente complementari, infatti la loro combinazione all'interno di una rete può portare ad un enorme valore aggiunto. Ad esempio, è possibile implementare un controllore SDN come una SFC in ambiente virtualizzato. Ciò significa che è possibile realizzare tramite l'utilizzo delle VNF il controllo centralizzato e la gestione degli applicativi usati da SDN (e.g. load balancing, rilevazione e analisi del traffico di dati). Allo stesso modo, SDN può facilitare la diffusione di NFV offrendo un modo flessibile e automatico per il concatenamento delle funzioni. In ogni caso, SDN e NFV restano concetti differenti accomunati dallo scopo di ottenere una *Software-driven networking solution*, ossia una rete basata sul software. Un'importante differenza è che mentre NFV può essere adattata su una rete pre-esistente, le SDN necessitano di una nuova rete dove il control plane ed il data plane siano separati.

2.2 Definizione e importanza del Firewall

I firewall sono dispositivi hardware o realizzazioni software che hanno come scopo quello di controllare il flusso di dati tra reti a diverso livello di sicurezza [10] o nel caso di local/personal firewall fornire protezione al nodo su cui sono installati (tipicamente un packet filter). Attraverso un insieme di regole opportunamente configurate sui firewall, questi permettono o negano il flusso di dati tra reti, organizzate in vario modo a seconda dei bisogni aziendali o del singolo utente. I firewall possono essere utilizzati sul perimetro di una rete, in cui si va a separare una rete considerata sicura, da una rete considerata non sicura (e.g. Internet), oppure possono essere posti all'interno della rete stessa per fornire protezione a risorse ritenute critiche (e.g. Server, Database). Una classica architettura di rete, quando si utilizza un firewall perimetrale, è quella mostrata in figura 2.10, in cui si vanno a distinguere tre zone diverse. La rete interna è quella considerata sicura. In base all'organizzazione della rete, questa parte potrebbe essere suddivisa in ulteriori zone, a seconda del livello di sicurezza che ogni zona garantisce. La *De-Militarized Zone*

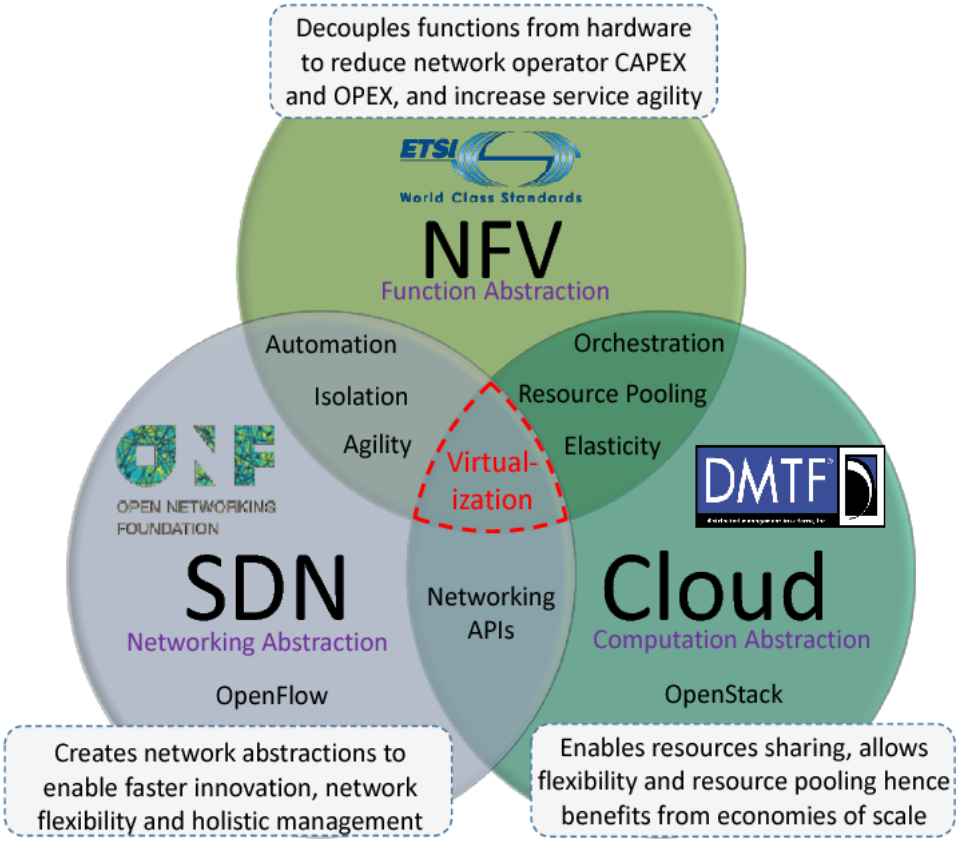

Figura 2.9. Relazione tra NFV, SDN e Cloud

(DMZ), viene considerata come zona neutrale in cui si vanno a collocare i server che hanno bisogno dell'accesso pubblico (e.g. Mail Server, Web Server), in modo che se si dovesse verificare un attacco ad un server nella DMZ, la rete interna non viene direttamente compromessa, mantenendo il suo stato di sicurezza. La zona non sicura viene identificata come la parte di rete che si affaccia direttamente su Internet, in cui non è presente nessun controllo sul flusso di dati in arrivo, e zona da cui potrebbero provenire la maggior parte dei tentativi di attacco.

Considerando il modello OSI, mostrato in Fig. 2.11, si può effettuare una classificazione dei firewall in base al livello di applicazione:

- **Packet filter stateless:** agisce al livello 3 e 4 del modello OSI, ed effettua filtraggio dei pacchetti basandosi sulle informazioni contenute nell'intestazione IP, come l'indirizzo sorgente/destinazione, e informazioni sui protocolli *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP), come la porta sorgente/destinazione. Questa tipologia viene indicata anche con il nome *stateless inspection firewall*.
- **Packet filter stateful:** fornisce le stesse funzionalità del caso precedente, migliorandone le prestazioni, in quanto tiene conto dello stato della connessione. Questa tipologia viene indicata anche con il nome *stateful inspection firewall*.
- **Application firewall:** agisce ai livelli 3, 4 e 7 del modello OSI e fornisce protezione ad applicazioni e servizi. Confronta il comportamento dei protocolli con comportamenti definiti in standard da seguire, prevenendo cattivi utilizzi del protocollo stesso. Effettua anche controlli sui protocolli dichiarati, analizzandone il contenuto per evitare attacchi che utilizzano il tunnel.
- **Application-proxy gateway firewall:** agisce a tutti i livelli del modello OSI, definendo un proxy che agisce come intermediario tra due host che comunicano. Può implementare funzioni di autenticazione per gli utenti della rete. Differisce dall'application firewall in

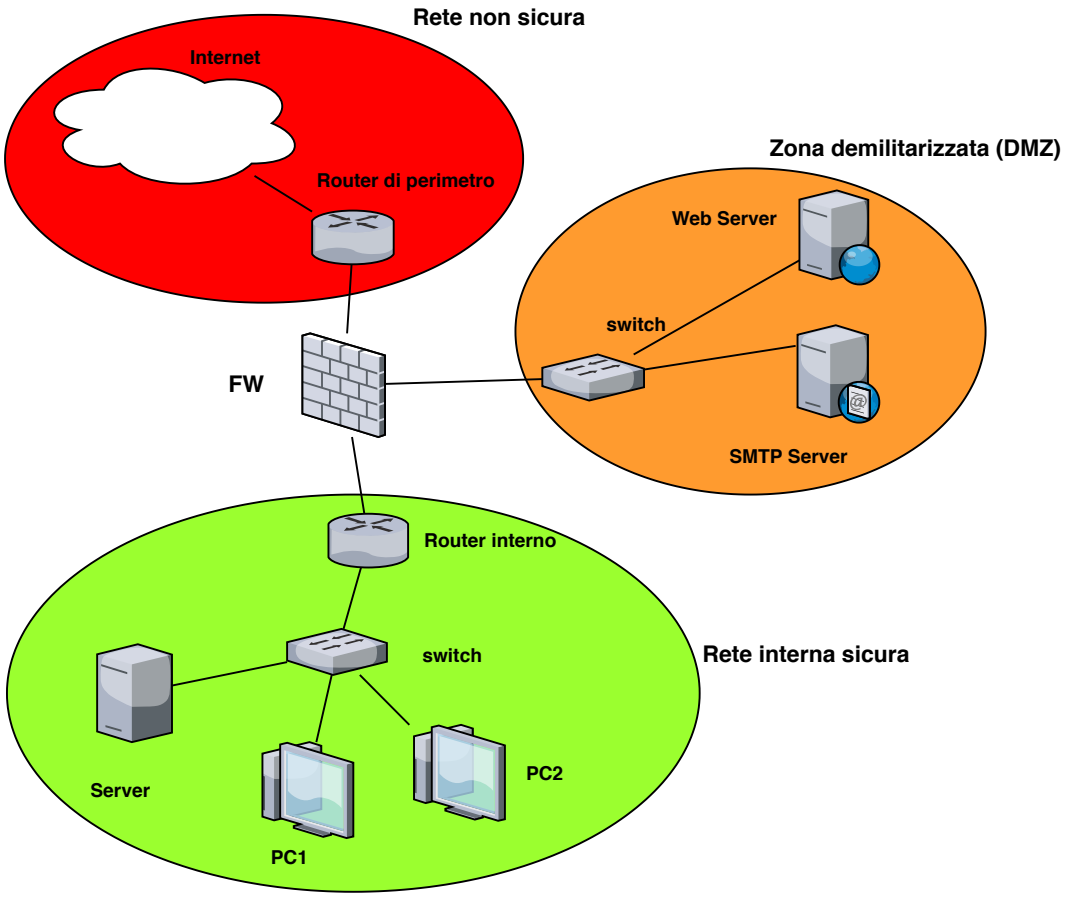

Figura 2.10. Tipica topologia di rete con firewall perimetrale

quanto previene una connessione diretta tra due host. Inoltre un application-proxy gateway può decifrare i pacchetti, esaminarli e re-inviarli cifrati. Fornisce il livello di sicurezza più alto, a discapito delle prestazioni, in quanto l'analisi effettuata sui pacchetti risulta più dispendiosa.

Nel contesto della tesi si va a realizzare una vNSF che vada a fornire le funzionalità di *packet filter stateful*. Il vantaggio principale di questa tipologia di firewall risulta essere la velocità di analisi dei pacchetti e l'indipendenza dalle applicazioni, in quanto il contenuto del pacchetto non viene analizzato. Tuttavia esso presenta alcune debolezze note:

- Non riesce a prevenire attacchi che sfruttano vulnerabilità applicative. Il traffico di un'applicazione può essere consentito o bloccato, ma non si ha controllo sui comandi che l'applicazione esegue.
- Scarse informazioni fornite per funzionalità di *logging*, le quali sono spesso coincidenti con gli stessi campi utilizzati per definire le regole (e.g. indirizzi, porte).
- Difficile eseguire l'autenticazione degli utenti.
- Vulnerabile ad attacchi che alterano le informazioni contenute nel livello 3 del modello OSI (e.g. IP Spoofing).
- Configurazione complessa.

Per la realizzazione della vNSF vengono analizzate le tecnologie Netfilter/Iptables [11] e *Packet Filter* (PF) [12], identificate come due soluzioni Open-Source con caratteristiche che rispecchiano

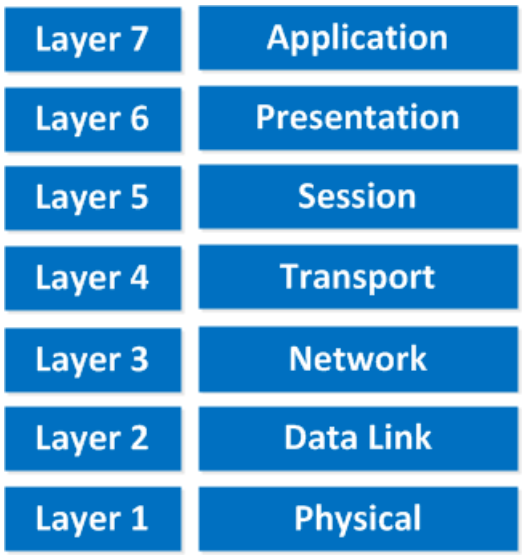

Figura 2.11. Modello OSI

le funzionalità che si vogliono implementare nella vNSF, oltre ad essere due soluzioni largamente utilizzate. A testimonianza dell'affidabilità e delle buone performance che queste due soluzioni riescono a garantire, essi forniscono il modulo di packet filtering per diverse applicazioni firewall con caratteristiche avanzate ampiamente utilizzate (e.g. PfSense [13], IPFire [14]).

2.2.1 Netfilter/Iptables

Netfilter/Iptables fornisce un'implementazione di packet filter Open Source utilizzabile su piattaforma Linux a partire dalla versione kernel 2.4.x in poi. Grazie alle funzionalità che mette a disposizione fornisce una valida alternativa ai packet filter commerciali. Netfilter/Iptables viene spesso riferito come componente unico, ma esso è composto da due moduli: Netfilter e Iptables. Iptables fornisce l'interfaccia di gestione del firewall Linux e rappresenta la versione successiva e migliorata dei vecchi sistemi ipchains, utilizzato sui sistemi Linux 2.2.x, ed ipfwadm, risalente ai sistemi Linux 2.0.x. Netfilter è il componente che realizza le funzionalità di filtraggio nello spazio kernel Linux. Esso processa i pacchetti in base alle informazioni nell'intestazione IP, indirizzo sorgente/destinazione, porte e protocollo. Netfilter fornisce inoltre supporto per l'ispezione stateful dei pacchetti (grazie al modulo *conntrack*) ed altre caratteristiche di base utili per un packet filter, il tutto configurabile tramite *Command Line Interface* (CLI) [17]. Sono supportate anche le funzionalità di *Network Address Translation* (NAT), ossia la traduzione degli indirizzi di rete, distinguibile in *Source NAT* (SNAT), se viene modificato l'indirizzo sorgente del pacchetto, e *Destination NAT* (DNAT) se invece viene modificato l'indirizzo destinazione.

Iptables al fine di ottenere una migliore organizzazione delle politiche impostate utilizza il concetto di tabelle. Le regole vengono raggruppate in tabelle differenti in base al tipo di decisione per cui vengono utilizzate. Le tabelle sono:

- **Filter Table:** è la tabella utilizzata per effettuare le decisioni di filtraggio, ossia quelle regole che decidono se lasciar passare il pacchetto o scartarlo.
- **NAT Table:** è la tabella utilizzata per implementare le funzioni di traduzione degli indirizzi, ossia le regole che determinano quando ed in che modo modificare l'indirizzo sorgente o destinazione del pacchetto.
- **Mangle Table:** è la tabella utilizzata per modificare l'intestazione IP dei pacchetti. Ad esempio è possibile modificare il *Time To Live* (TTL) dell'header IP, inteso come numero di router che il pacchetto può attraversare prima di essere scartato.

- **Raw Table:** è la tabella utilizzata per definire i pacchetti per cui non si deve tener traccia dello stato della connessione, in quanto Netfilter/Iptables fornisce di base funzionalità di filtraggio *stateful*.

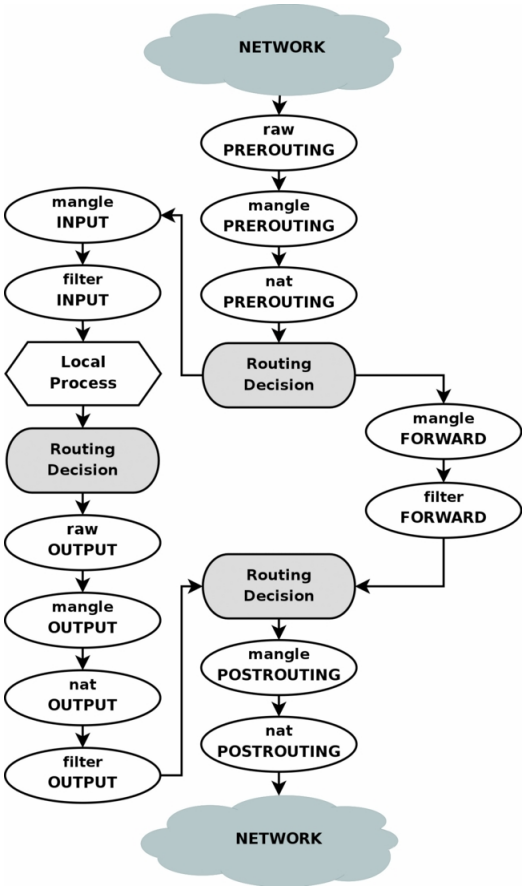

Figura 2.12. Netfilter/Iptables, mappa di attraversamento.

All'interno delle tabelle, le regole sono a loro volta organizzate in catene che forniscono un ordine di attraversamento dei pacchetti, ed ogni pacchetto viene processato secondo la figura 2.12. In base alla catena di appartenenza di una politica, si determina in quale momento essa debba essere presa in considerazione. Le catene sono:

- **PREROUTING:** questa catena viene innescata dal traffico in ingresso, prima che venga presa qualunque decisione sul routing del pacchetto. È presente nelle tabelle raw, mangle, nat (DNAT).
- **INPUT:** questa catena viene innescata da un pacchetto in ingresso, dopo che la decisione di routing è stata presa e se il pacchetto è destinato al sistema locale. È presente nelle tabelle mangle, filter, nat (SNAT).
- **FORWARD:** questa catena viene innescata dopo che la decisione di routing su un pacchetto è stata presa, se il pacchetto deve essere inviato ad un altro host. È presente nelle tabelle mangle, filter.
- **OUTPUT:** questa catena viene innescata per il traffico generato localmente, inviato sulla rete. È presente nelle tabelle raw, mangle, nat(DNAT), filter.
- **POSTROUTING:** questa catena viene invocata per tutto il traffico in uscita, che sia generato localmente o inoltrato verso altri host, poco prima di essere inviato. È presente nelle tabelle mangle, nat (SNAT).

Oltre alle catene descritte, Iptables permette all'amministratore di creare delle catene *user-defined*. Attraverso la definizione di politiche specifiche, è possibile "saltare" dal modulo Netfilter allo spazio utente in cui vengono eseguite le catene personalizzate. Ciò fornisce la possibilità di specificare regole *ad-hoc* per casi specifici. D'altra parte, uscendo dallo spazio kernel ed effettuando operazioni nello spazio utente del sistema si ottiene un degrado di performance del packet filter.

2.2.2 PF

Packet Filter, spesso riferito con il solo termine PF, nasce con OpenBSD nel 2001, come sostituto di IPFilter, a causa di problemi di licenza. A partire da quel momento viene considerato uno degli strumenti open-source per implementare funzionalità di *firewalling* e gestione del traffico più potenti [18]. La caratteristica principale di PF è quella di filtrare il traffico analizzando i pacchetti ricevuti, basandosi su:

- indirizzo sorgente e destinazione;
- protocollo;
- porta sorgente e destinazione;
- interfaccia;
- direzione del pacchetto;

PF supporta l'elaborazione *stateful* dei pacchetti, conservando le informazioni sullo stato delle connessioni di rete in una *state table*. Grazie alla tabella degli stati PF riesce a verificare velocemente se un pacchetto fa parte di una connessione stabilita, evitando di processare le regole impostate. Questo porta un miglioramento alle performance di filtraggio dei pacchetti. Risulta in ogni caso possibile definire regole per cui non si deve tenere traccia dello stato.

Oltre alle classiche opzioni di filtraggio sui pacchetti, proprie di un packet filter, PF fornisce altre opzioni utili, come quelle introdotte dalle politiche di **scrub** e **antispoof**. La prima abilita la normalizzazione del traffico che corrisponde alla regola, riassemblando i pacchetti frammentati e permettendo così di evitare gli attacchi dovuti ad un'errata gestione della frammentazione del traffico. La seconda effettua un controllo sugli indirizzi provenienti da un'interfaccia, bloccando tutti i pacchetti contenenti informazioni che sono logicamente incompatibili con la rete collegata su tale interfaccia. Questo controllo è possibile solo quando si è a conoscenza degli indirizzi attendibili dal collegamento con una rete e non costituisce una soluzione completa per gli attacchi di *IP Spoofing*, a cui il packet filter resta vulnerabile.

PF viene spesso utilizzato con il *Common Address Redundancy Protocol* (CARP) e pfsync. CARP permette la condivisione dello stesso indirizzo IP ad un gruppo di host sullo stesso segmento di rete, tale gruppo prende il nome di "redundancy group" ed è composto da un nodo master e da nodi di backup. Pfsync fornisce invece uno strumento per la sincronizzazione della *state table*. I cambiamenti effettuati sulla tabella degli stati di PF vengono monitorati ed esposti sull'interfaccia di rete in modo che gli altri nodi su cui PF è in esecuzione possano sincronizzarsi con tali cambiamenti. L'utilizzo di PF combinato con CARP e pfsync permette di creare un cluster di packet filter altamente affidabile e ridondante.

Pur essendo nativo di OpenBSD, per gli scopi della tesi è stato scelto il sistema operativo FreeBSD, in quanto riesce a garantire delle prestazioni di rete migliori [19], ed un miglior supporto per gli strumenti utilizzati durante lo sviluppo della vNSF.

2.2.3 Analisi di performance tra firewall open-source

In questa sezione vengono messe a confronto le caratteristiche di PF e di Netfilter/Iptables. Si possono evidenziare alcune differenze a livello di di funzionamento:

- **Elaborazione delle regole:** PF confronta i pacchetti ricevuti con le regole contenute nel file di configurazione. Anche se un pacchetto rispetta una regola, il comportamento di base è effettuare il confronto con le altre regole nel file mantenendo come politica da adottare quella specificata nell'ultima regola trovata. Se però il pacchetto corrisponde ad una regola definita come *quick*, tale regola verrà subito scelta come politica da adottare, senza che il pacchetto venga confrontato con altre regole. Netfilter/iptables confronta i pacchetti seguendo la struttura delle tabelle e delle catene descritte in Fig. 2.12, eseguendo l'azione indicata nella prima regola che viene rispettata dal pacchetto.

- **Filtraggio dei pacchetti:** PF fornisce tutte le caratteristiche di base di un packet filter, introducendo alcune opzioni avanzate, come quelle di antispoof e rilevamento passivo del sistema operativo. Netfilter/Iptables permette di definire dei moduli *ad-hoc* per il filtraggio, permettendo di specificare delle catene in grado di rispondere a casi specifici, seguendo criteri personalizzati. Moduli che però effettuano le operazioni di filtraggio nello spazio utente, abbandonando lo spazio kernel, ed influendo in modo negativo sulle performance del packet filter. Inoltre a differenza di PF permette di effettuare filtraggio sullo stato attuale del pacchetto analizzato.

- **Prestazioni:** PF pur non permettendo un filtraggio basato sullo stato dei pacchetti, fornisce una soluzione *stateful* in quanto tiene conto dello stato della connessione lasciando passare quelli che fanno parte di una comunicazione già stabilita senza confrontarli con le regole definite. Ciò è ottenuto definendo al suo interno una tabella ordinata, in cui viene effettuata una ricerca efficiente sulle connessioni già stabilite. Netfilter/Iptables effettua in ogni caso il confronto con le regole contenute al suo interno. Ciò influisce in modo negativo sulle prestazioni del packet filter, soprattutto nel caso di ruleset complessi o numerosi.

Dopo un'analisi sulle differenze a livello di funzionamento dei packet filter, viene realizzato un test in cui si vanno a monitorare le loro prestazioni al variare della mole di traffico da gestire e del numero di regole impostate su di essi. Gli strumenti utilizzati per l'analisi di performance sono i seguenti:

- **Nginx [20]:** fornisce una implementazione leggera ma ad alte prestazioni di un server software HTTP e di reverse proxy server. Nginx può essere eseguito su sistemi Unix, Linux e Microsoft. Esso è famoso per la sua stabilità, per la flessibilità della sua configurazione e per il basso consumo di risorse, pur riuscendo a garantire prestazioni elevate. Infatti se comparato con Apache, Nginx riesce a fornire supporto a più connessioni concorrenti (circa 50,000). Viene utilizzato su una delle macchine da test per implementare un server HTTP, protetto dal packet filter su cui si andranno a valutare le prestazioni.

- **ApacheBench (ab) [16]:** fornisce uno strumento distribuito da Apache per la valutazione di un web server HTTP. Esso permette di specificare parametri di test, quali il numero di connessioni concorrenti da avviare, o il numero di richieste totali da effettuare, in modo da valutare le prestazioni di rete con carichi di lavoro diversi ed ottenere un benchmark preciso dal lato dell'end-user [21]. Viene utilizzato sulla macchina client per effettuare richieste verso il server web HTTP ed effettuare misurazioni sull'impatto del packet filter.

- **Kernel-based Virtual Machine (KVM) [15]:** fornisce una soluzione open-source di virtualizzazione completa per i sistemi Linux su hardware x86. Utilizzando KVM si possono eseguire macchine virtuali, ognuna con un proprio hardware virtualizzato (e.g. scheda di rete, disco, adattatore grafico, etc..).

Per l'esecuzione del test viene realizzata una connessione punto-punto, mostrata in Fig. 2.13, per ridurre al minimo i ritardi dovuti alla propagazione del pacchetto e misurare i ritardi dovuti alle prestazioni del packet filter sotto analisi. Per l'esecuzione del test viene utilizzato KVM sul PC2 in quanto si vuole eseguire il packet filter in ambiente virtualizzato, per andare a simulare l'ambiente in cui viene eseguita una Virtual Network Function.

Le macchine PC1 e PC3 vengono assegnate a due sottoreti differenti, utilizzando per la sottorete che collega PC1 con l'interfaccia `eth0` di PC2 la subnet 192.168.2.0/24, mentre per il

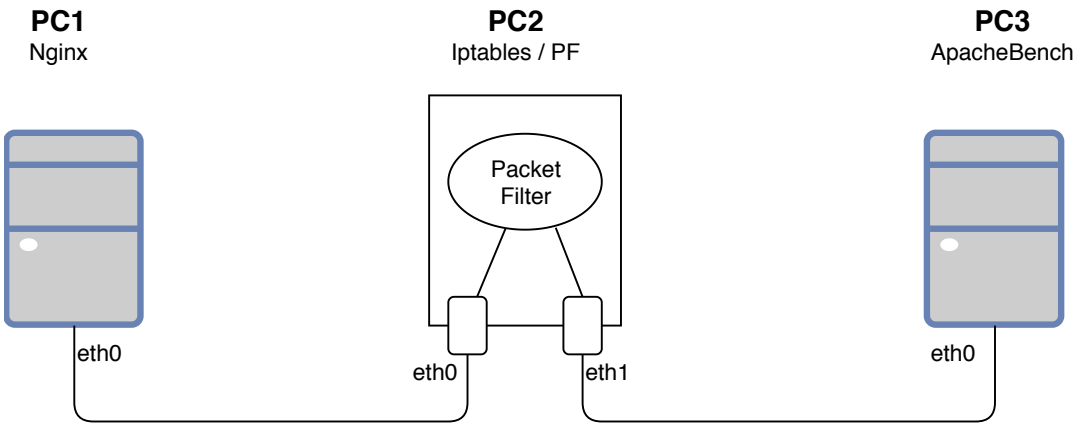

Figura 2.13. Architettura utilizzata per il test

collegamento tra `eth1` ed `eth0` del PC3 vengono assegnati gli indirizzi della subnet 192.168.1.0/24. La macchina virtuale viene collegata utilizzando dei bridge sulle interfacce di rete della macchina host. Come sistema operativo per il test di Iptables viene utilizzato Alpine Linux 3.7. Esso è realizzato per essere leggero ed efficiente, ed è stato ritenuto ideale per testare le prestazioni del packet filter evitando che le performance fossero degradate da altri processi "pesanti" del sistema operativo. Per il test di PF viene scelto FreeBSD 11.1. Come anticipato, pur essendo PF nativo di OpenBSD, vengono garantite prestazioni di rete migliori su FreeBSD. Le configurazioni delle macchine utilizzate per il test sono fornite nella tabella 2.1.

Dispositivo	**CPU**	**Memoria**	**Scheda di rete**	**OS**
PC1	Intel(R) Core(TM) 2 Duo 2.13GHz 2MB L2 CACHE	RAM 2GB DDR2	NIC Gigabit[1] Ethernet Intel 825444G	CentOS 7
PC2	Intel(R) Core(TM) 2 Duo 2.13GHz 2MB L2 CACHE	RAM 2GB DDR2	NIC Gigabit[1] Ethernet Intel 825444G	CentOS 7
PC3	Intel(R) Core(TM) i7-3632QM 2.20GHz 6MB L3 CACHE	RAM 8GB DDR3	QCA8172 Fast Ethernet[2] Qualcomm Atheros	Ubuntu 16.04 LTS

Tabella 2.1. Configurazione macchine

Il test viene eseguito utilizzando ApacheBench per inviare pacchetti verso il PC3 su cui è in esecuzione Nginx, utilizzando un numero variabile di sessioni concorrenti per introdurre un diverso carico di lavoro. Si va inoltre ad utilizzare diverse configurazioni del packet filter per monitorare le prestazioni al variare del numero di regole da confrontare per ogni pacchetto ricevuto. Ogni test viene eseguito tre volte per evitare risultati dovuti a casi eccezionali nella rete, viene mostrato come risultato la mediana dei tre test. I risultati vengono mostrati in Tabella 2.2, Tabella 2.3 e

[1] Rate massimo supportato di 1000 Mbit/s

[2] Rate massimo supportato di 100 Mbit/s

Tabella 2.4. Si vanno a confrontare nella singola tabella le prestazioni al variare del numero di regole con il numero di sessioni concorrenti fissato, nelle tabelle Netfilter/Iptables viene riferito come Iptables.

# Regole	Richieste al secondo		Tempo per richiesta (ms)		Rate trasferimento (kB/s)	
	Iptables	**PF**	**Iptables**	**PF**	**Iptables**	**PF**
0	2747.54	2732.99	90.990	91.475	10555.57	10499.58
500	2169.75	2624.45	115.221	95.258	8335.73	10082.60
1000	1452.80	2624.30	172.082	95.263	5581.35	10082.03
1500	1228.80	2418.93	203.540	103.351	4720.81	9293.05
2000	882.64	1318.72	283.242	189.578	3391.00	5066.26
2500	738.89	846.26	338.347	295.418	2838.65	3251.80
3000	646.17	700.61	386.893	356.829	2482.50	2691.97
3500	552.41	639.74	452.559	390.789	2122.26	2458.31
4000	482.70	555.73	517.920	449.857	1854.44	2135.78
4500	422.22	504.64	592.113	495.406	1622.07	1939.09

Tabella 2.2. Confronto con 250 sessioni concorrenti

# Regole	Richieste al secondo		Tempo per richiesta (ms)		Rate trasferimento (kB/s)	
	Iptables	**PF**	**Iptables**	**PF**	**Iptables**	**PF**
0	2765.82	2743.86	180.778	182.225	10625.72	10541.33
500	2290.17	2597.13	218.324	192.521	8798.38	9977.63
1000	1385.63	2586.49	359.809	193.312	5338.66	9936.76
1500	1133.28	2475.22	441.199	202.002	4353.82	9509.30
2000	848.94	1273.78	588.969	392.533	3261.84	4893.59
2500	710.11	844.83	704.121	591.835	2728.36	3247.01
3000	608.17	720.48	822.135	693.984	2728.36	2768.84
3500	528.99	636.37	945.194	786.704	2032.62	2445.79
4000	459.76	556.70	1087.532	898.148	1766.57	2140.08
4500	426.68	500.30	1171.825	999.401	1639.93	1923.18

Tabella 2.3. Confronto con 500 sessioni concorrenti

Dalle tabelle si può notare che le prestazioni di rete, in assenza di regole impostate sul packet filter, risultano migliori sul sistema operativo Alpine. Con l'aumentare del numero di regole però, le prestazioni risultano migliori utilizzando PF, differenza di performance che risulta più marcata al raggiungimento delle 1000 regole. Le prestazioni di PF subiscono un calo netto al raggiungimento delle 2000 regole, ma restano comunque migliori di quelle di Netfilter/Iptables, anche se in modo meno marcato.

Considerando l'analisi sul funzionamento ed il test sulle performance risulta che Iptables offre caratteristiche avanzate nella definizione delle regole di filtraggio, considerate però superflue ai fini delle caratteristiche richieste al contesto di utilizzo del vNSF che si intende realizzare. PF riesce a garantire le funzionalità richieste ad una vNSF con funzioni di packet filter. Inoltre PF si è dimostrato più performante di Iptables per carichi di lavoro crescenti, che possono risultare frequenti nell'ambito di funzionamento di una vNSF. Con il test realizzato dunque, si è scelto di utilizzare PF come tecnologia per la realizzazione della vNSF.

2.3 Configurazione delle funzioni di rete basata su policy

Nell'ambiente cloud un aspetto fondamentale da prendere in considerazione riguarda la configurazione delle funzioni di rete. Un errore nella configurazione delle funzionalià di rete può risultare in falle

# Regole	Richieste al secondo		Tempo per richiesta (ms)		Rate trasferimento (kB/s)	
	Iptables	**PF**	**Iptables**	**PF**	**Iptables**	**PF**
0	2705.96	2670.19	369.554	347.505	10395.76	10258.33
500	2014.84	2553.48	496.318	391.622	7740.60	9809.95
1000	1304.12	2554.83	766.802	391.415	5010.15	9815.15
1500	1047.14	2025.16	954.980	493.789	4024.06	7780.24
2000	874.13	1194.26	1143.994	837.341	3359.42	4588.09
2500	702.92	843.86	1422.641	1185.029	2701.60	3243.99
3000	602.03	710.44	1661.045	1407.580	2314.36	2731.99
3500	547.42	626.31	1826.753	1596.666	2104.30	2409.66
4000	457.31	539.49	2186.708	1853.618	1758.80	2075.42
4500	389.41	489.13	2568.010	2044.434	1497.21	1881.78

Tabella 2.4. Confronto con 1000 sessioni concorrenti

di sicurezza pronte ad essere sfruttate dagli attaccanti, con conseguenti danni dovuti ad accessi a dati sensibili. Per questo motivo quando viene effettuata una configurazione, oltre ai problemi legati ad efficienza e performance, bisogna prestare attenzione anche agli aspetti di sicurezza. Tuttavia la grande varietà di implementazioni relative ad uno stesso servizio rende ancor più complicata la creazione di un protocollo di sicurezza. A tal proposito vengono studiate soluzioni basate su una architettura *Policy-Based Management* (PBM). L'idea di questo tipo di gestione è fornire un'astrazione per le regole di configurazione atte a descrivere il comportamento di un sistema. Queste regole possono essere utilizzate per verificare che il sistema soddisfi i requisiti di sicurezza. Andremo in seguito ad approfondire i concetti che permettono una gestione basata su policy di sicurezza.

2.3.1 Concetti di base

L'*Internet Engineering Task Force* (IETF) nel *Request For Comments* (RFC) 3198 [22] definisce formalmente alcuni concetti fondamentali nel contesto PBM. Il concetto principale è quello di policy, definito secondo due prospettive:

1. *"un obiettivo definito, percorso o metodo di azione per guidare o determinare le decisioni presenti e future. Le policy sono implementate o eseguite in un contesto particolare (come le policy definite in un'unità aziendale)"*;
2. *"le policy come un insieme di regole per amministrare, gestire e controllare l'accesso alle risorse di rete"*;

Nella definizione e applicazione di una policy si possono individuare due figure principali:

- **Subject:** fornisce l'entità o l'insieme di entità che origina una richiesta e che viene autorizzata o non autorizzata ad eseguire tale richiesta.
- **Target:** rappresenta l'entità o l'insieme di entità affette da una policy. Ad esempio, i target di una policy che riconfigura un dispositivo di rete sono i singoli servizi che vengono configurati.

Oltre al concetto centrale di policy, nell'RFC 3198 vengono definiti altri elementi comuni al lessico di una gestione basata su policy, come:

- **Policy rule:** rappresenta il blocco di base di un sistema PBM. Una policy rule si presenta nella forma *Event-Condition-Action* (ECA), ossia risponde alla regola *"quando un Event si verifica, se la Condition è verificata, esegui l'Action"*. La struttura di una Policy rule è mostrata in Fig. 2.14.

- **Policy event:** rappresenta qualunque importante occorrenza nel tempo che sia relativa ad un cambiamento del sistema gestito. Un evento viene utilizzato per stabilire quali policy condition debbano essere valutate in quel momento. Alcuni esempi di evento sono dati dalle operazioni svolte dagli utenti di un sistema (e.g. logon, logoff).
- **Policy condition:** rappresenta lo stato necessario e i pre-requisiti da soddisfare affinchè una policy rule venga eseguita. Quando la condizione è soddisfatta, la policy rule può essere applicata, a meno di altri fattori di priorità o strategie decisionali.
- **Policy action:** definisce quali operazioni debbano essere svolte dalla policy rule, quando le condizioni per applicarla sono soddisfatte. Queste azioni possono influenzare il flusso del traffico o la configurazione delle risorse di rete.
- **Policy group:** rappresenta un contenitore in cui vengono aggregate le policy rule o altri policy group. Permette il raggruppamento delle regole in una Policy.

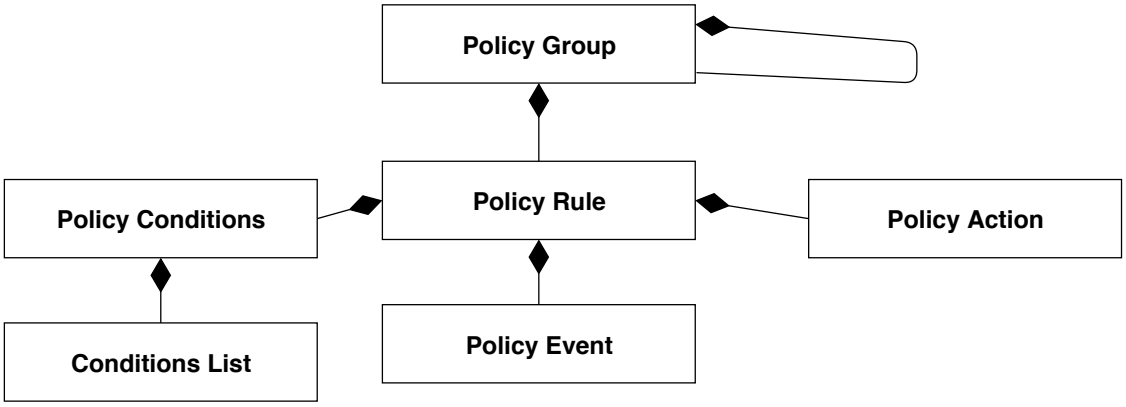

Figura 2.14. Struttura di una policy rule

2.3.2 Architettura PBM

L'IETF propone, nell'RFC 3060 [23], un'architettura base per il PBM, proposta in Fig. 2.15. Si possono identificare quattro elementi principali:

- **Policy Management Tool (PMT):** rappresenta l'entità che permette all'utente di specificare le policy di sicurezza della rete. La policy è composta da una o più policy rule, specificate con un linguaggio di livello più basso rispetto a quello utilizzato dall'utente (vedi sezione 2.3.3).
- **Policy Repository (PR):** rappresenta l'entità in cui vengono memorizzate le policy create dal PMT.
- **Policy Decision Point (PDP):** rappresenta il modulo di decisione dell'architettura. Viene notificato dal PMT ogni volta che avviene una modifica nel PR. Il PDP effettua la lettura della policy dal PR e prende le decisioni in base ai requisiti impostati.
- **Policy Enforcement Point (PEP):** rappresenta l'entità su cui le policy, in seguito alle decisioni prese dal PDP, vengono realmente applicate.

La maggior parte delle interazioni tra i componenti individuati avviene in modo automatico all'interno dell'architettura. L'utente ha il solo compito di specificare le politiche ad alto livello. Con questo approccio tutta la complessità della configurazione del sistema viene spostata a livello di specifica della policy. Se essa viene specificata in modo errato o entra in conflitto con regole precedentemente definite si può verificare un'errore nelle decisioni prese dal PDP, che a questo punto potrebbe introdurre vulnerabilità sfruttabili dagli attaccanti o bloccare il traffico di rete

che dovrebbe essere consentito. Risulta dunque essenziale definire un linguaggio ad alto livello che gli utenti possano sfruttare per la specifica delle policy di rete, un modulo di traduzione delle regole ed un modulo per l'individuazione di conflitti o anomalie sulle policy specificate.

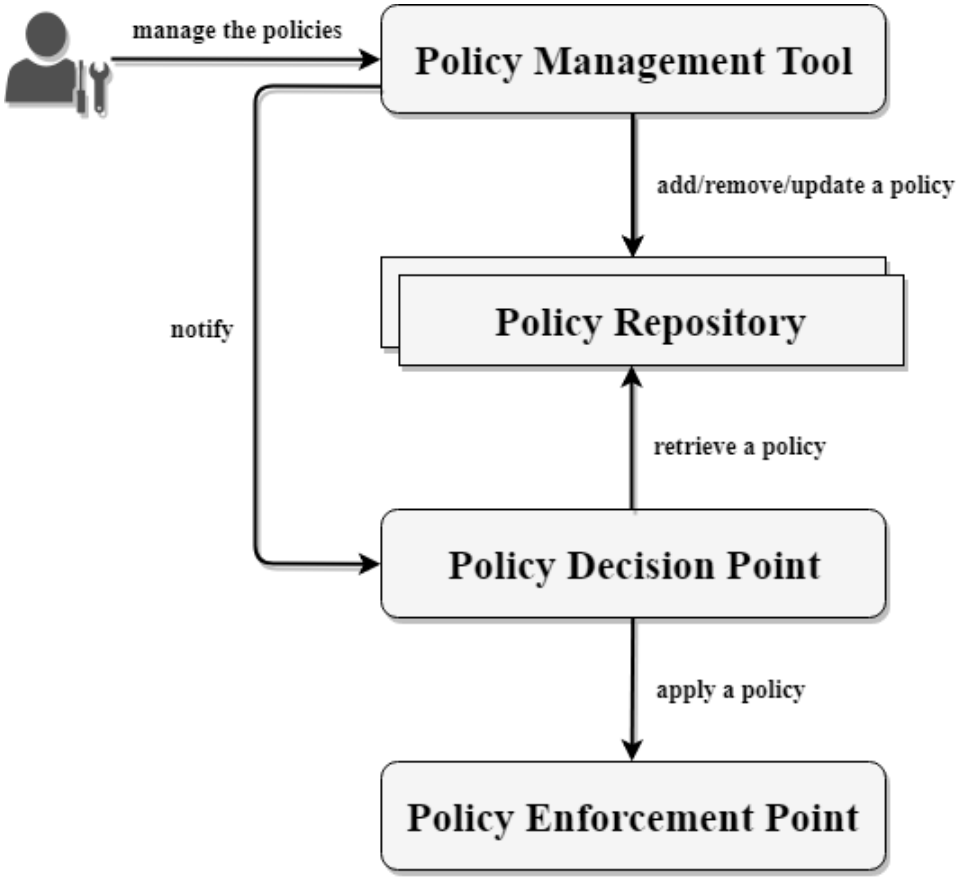

Figura 2.15. Architettura PBM di base proposta dall'IETF

2.3.3 Astrazione e specifica delle policy

Le policy di sicurezza possono essere specificate a tre diversi livelli:

- **Policy di alto livello:** si specifica con una sintassi altamente comprensibile all'utente che prevede costrutti del linguaggio comune. In questo modo l'utente che configura il sistema non deve essere esperto dell'implementazione delle singole funzionalità nel sistema. Un esempio di policy ad alto livello è data da "Abilita il traffico web".
- **Policy di medio livello:** vengono tradotte dalle politiche ad alto livello per raggiungere un formato intermedio. In questo stadio la specifica della policy risulta indipendente dall'implementazione della funzione di rete da configurare. Ad esempio si passa da una policy di alto livello del tipo "Abilita il traffico web" ad una policy di medio livello "Abilita le connessioni TCP sulla porta 80".
- **Policy di basso livello:** rappresenta la configurazione reale della funzione di rete che risulta dalla traduzione della policy di medio livello. Questo livello è dipendente dall'implementazione scelta per la realizzazione della funzionalità.

Per la rappresentazione delle policy, nell'articolo [24], viene presentato un approccio che prevede l'utilizzo di due linguaggi user-oriented per la specifica delle policy di sicurezza. I linguaggi proposti sono chiamati *High Security Policy Language* (HSPL) e *Medium Security Policy Language*

(MSPL). HSPL permette di specificare i requisiti del sistema con frasi nella forma "soggetto-verbo-oggetto-parametri", le quali possono essere facilmente comprese e composte da qualsiasi utente. Ogni termine del costrutto viene scelto da un insieme predefinito di possibilità. Ciò permette di non costringere l'utente ad imparare una nuova sintassi e di effettuare una mappatura precisa tra le scelte effettuate ed i componenti che formano il costrutto HSPL. Risulta però possibile per l'utente personalizzare la regola tramite l'utilizzo di parametri (e.g. specifico URL, vincoli sul tempo). MSPL fornisce un linguaggio indipendente dall'implementazione di un servizio, organizzato in base alla tipologia della funzione per cui deve essere applicata la policy (e.g. packet filter). Deve risultare un linguaggio flessibile ed estendibile in modo da supportare l'introduzione di nuove vNSF. In Fig. 2.16, viene mostrato il flusso di traduzione delle policy in configurazioni di basso livello. Il processo prevede due passi principali:

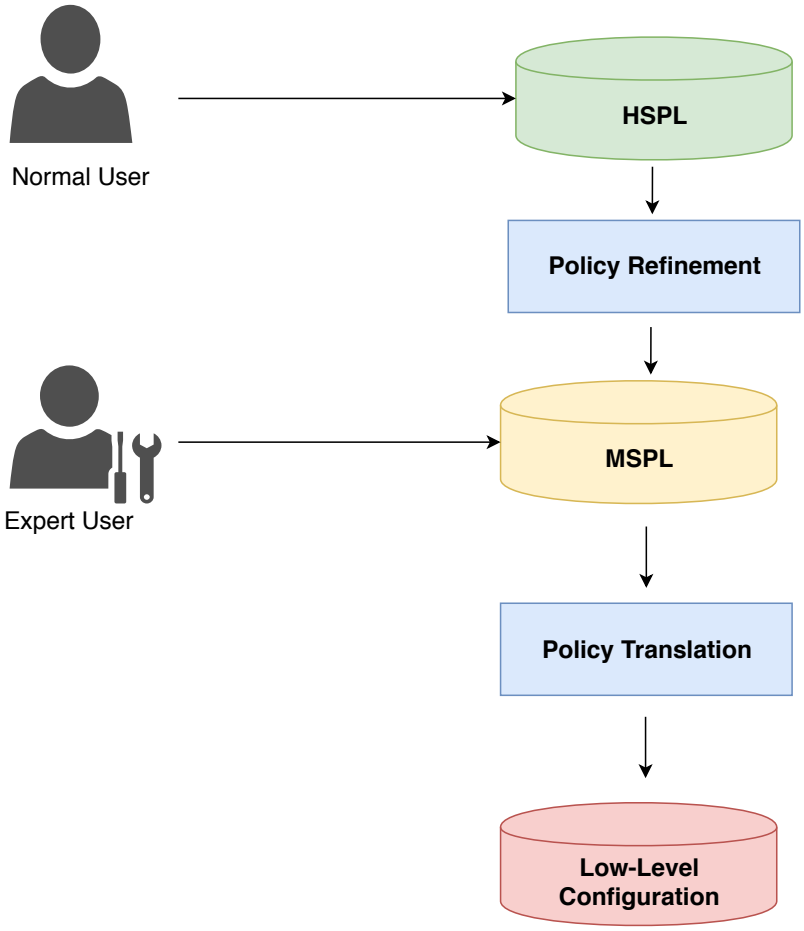

Figura 2.16. Flusso di trasformazione delle policy in configurazioni di basso livello

1. **Perfezionamento della policy:** questo processo risulta sofisticato poichè serve a determinare le risorse necessarie per soddisfare i requisiti di sistema per poter passare dalla specifica ad alto livello ad azioni da effettuare sul sistema (in figura *"policy refinement"*).
2. **Traduzione della policy:** questo processo risulta più semplice in quanto prevede un cambiamento di sintassi, in modo da adattare la policy indipendente dall'implementazione ad una configurazione specifica. (in figura *"policy translation"*).

2.4 Soluzione di PBM e vNSF nel progetto Europeo SHIELD

In questa sezione si contestualizzano le tecnologie introdotte, studiando il loro utilizzo nell'ambito del progetto Europeo SHIELD [25]. SHIELD utilizza le funzionalità introdotte da SDN/NFV

per creare dei livelli di protezione all'infrastruttura IT ricorrendo all'utilizzo delle vNSF. Nei paragrafi successivi si andrà ad analizzare come viene gestita la configurazione delle vNSF con l'utilizzo delle policy. Inoltre, si vedrà in che modo la vNSF debba essere adattata a tale contesto utilizzando come punto di riferimento l'implementazione di una funzione virtualizzata realizzante un packet filter basato sulla tecnologia Netfilter/Iptables e progettata per essere eseguita nello scenario SHIELD.

2.4.1 Topologia di sicurezza

Nel contesto del progetto SHIELD la definizione delle policy viene realizzata all'interno del DARE (vedi Fig. 2.6). Il DARE, oltre a contenere un modulo che provvede all'analisi dei dati, contiene il *Recommendation and Remediation Engine*, ossia un sotto-componente atto a definire e raccomandare azioni da effettuare in risposta a determinati eventi. Viene quindi definita una topologia di sicurezza in grado di fornire azioni di risposta ad una minaccia informatica [26]. Tali azioni, presentate all'utente tramite l'utilizzo di una Dashboard, si traducono in nuove istanze di vNSF configurate opportunamente, o applicazione di nuove configurazioni alle vNSF esistenti.

L'architettura del *Recommendation and Remediation Engine* viene mostrata in Fig. 2.17. Il sotto-componente ottiene informazioni sulla rete dalle vNSF in esecuzione sull'infrastruttura. Con i dati ottenuti ha un'ottima conoscenza dello stato della rete e riesce ad elaborare le azioni necessarie per garantire protezione dalle potenziali minacce. Le azioni vengono però proposte all'utente, a cui spetta il compito finale di prendere in considerazione o ignorare le soluzioni suggerite. Nel caso in cui tali operazioni vengano eseguite sarà il VNF Orchestrator a istanziare e configurare le vNSF necessarie. Viene di seguito fornita una descrizione dei blocchi che compongono il *Recommendation and Remediation Engine*.

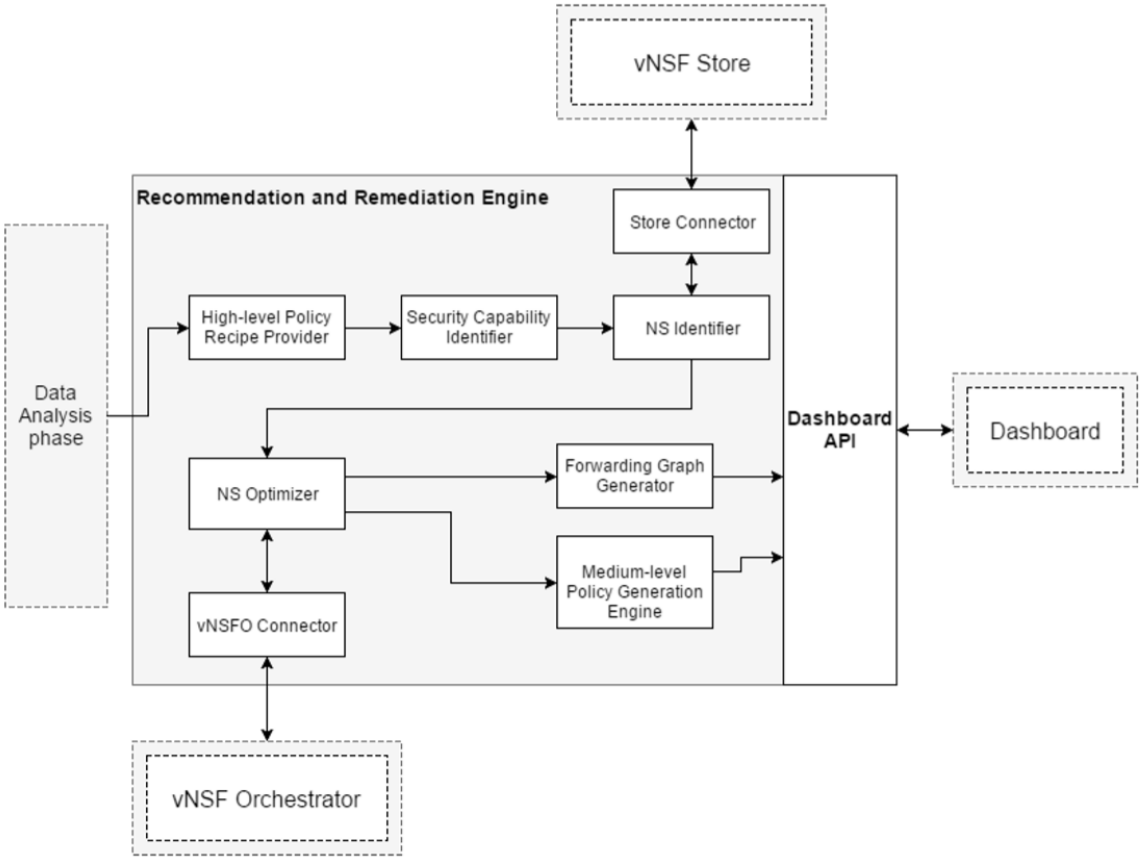

Figura 2.17. Architettura del *Recommendation and Remendation Engine*

High-level Policy Recipe Provider

Riceve in ingresso l'elaborazione dei dati eseguita dal modulo di analisi del DARE. I dati in ingresso forniscono informazioni relative alle minacce in atto sull'infrastruttura di rete. Il blocco in analisi provvede a definire e conservare in una repository interna le policy di sicurezza ad alto livello atte a contrastare le minacce informatiche presenti.

Security Capability Identifier

Crea una corrispondenza tra le policy ad alto livello fornite dall'*High-level Policy Recipe Provider* e le funzionalità di sicurezza necessarie a soddisfare i requisiti richiesti.

NS Identifier

Sfrutta le corrispondenze create dal *Security Capability Identifier* per selezionare l'insieme di vNSF (il NS) che soddisfanno tali requisiti. A tale scopo questo blocco deve interagire con il vNSF Store per avere conoscenza delle funzionalità fornite dalle vNSF che compongono i NS nel catalogo.

Store Connector

Gestisce il collegamento tra *NS Identifier* e il vNSF Store, fornendo una API per la richiesta di informazioni sui NS nel catalogo.

NS Optimizer

Seleziona un NS tra quelli proposti dal *NS Identifier* seguendo alcuni criteri di ottimizzazione. Inoltre verifica se un client sta già utilizzando il NS scelto in modo da aggiornare la configurazione del NS in esecuzione, evitando di creare due istanze dello stesso NS con due configurazioni differenti.

vNSFO Connector

Connette il *Recommendation and Remediation Engine* al *vNSF Orchestrator*. Fornisce un API tramite la quale è possibile ottenere informazioni sui NS in esecuzione.

Forwarding Graph Generator

Fornisce una descrizione topologica delle vNSF che permette al *vNSF Orchestrator* di effettuare il deploy delle vNSF sull'infrastruttura di rete.

Medium-level Policy Generation Engine

Genera le policy ad un livello medio di astrazione utilizzando le informazioni sulle funzionalità fornite dalle vNSF e da come esse sono distribuite sull'infrastruttura.

2.4.2 Implementazione del PBM e studio della vNSF

La gestione basata su policy in SHIELD utilizza i linguaggi HSPL e MSPL e segue il flusso di *policy refinement* e *policy translation* introdotto nella sezione 2.3.3. Tali linguaggi di specifica delle policy vengono utilizzati rispettivamente nel blocco *High-level Policy Recipe Provider*, in cui la policy ha un alto livello di astrazione, e nel modulo *Medium-level Policy Generation Engine*, in cui la policy presenta un livello di astrazione medio . La soluzione adottata in SHIELD ha come punto di partenza l'implementazione utilizzata per il progetto SECURED [27].

La soluzione implementata in SECURED prevede l'utilizzo di due moduli per la trasformazione delle policy:

- **H2M Service:** rappresenta il modulo che si occupa del *policy refinement*. Si occupa dunque della trasformazione da HSPL a MSPL.
- **M2L Service:** rappresenta il modulo che effettua l'operazione di *policy translation*, passando dalla policy specificata in MSPL alla configurazione specifica della vNSF.

Nell'implementazione SHIELD vengono sfruttate, all'interno del *Recommendation and Remediation Engine*, le funzionalità messe a disposizione dall'*H2M Service*, in modo da fornire la configurazione MSPL opportuna alle vNSF. Il modulo M2L Service non è invece presente all'interno del DARE, in quanto la traduzione da policy MSPL a configurazione locale viene implementata all'interno della vNSF stessa.

Viene ora presa in analisi una vNSF con funzionalità di packet filter realizzata nel contesto del progetto SHIELD e basata sulla tecnologia Netfilter/Iptables. Nella soluzione mostrata in Fig.

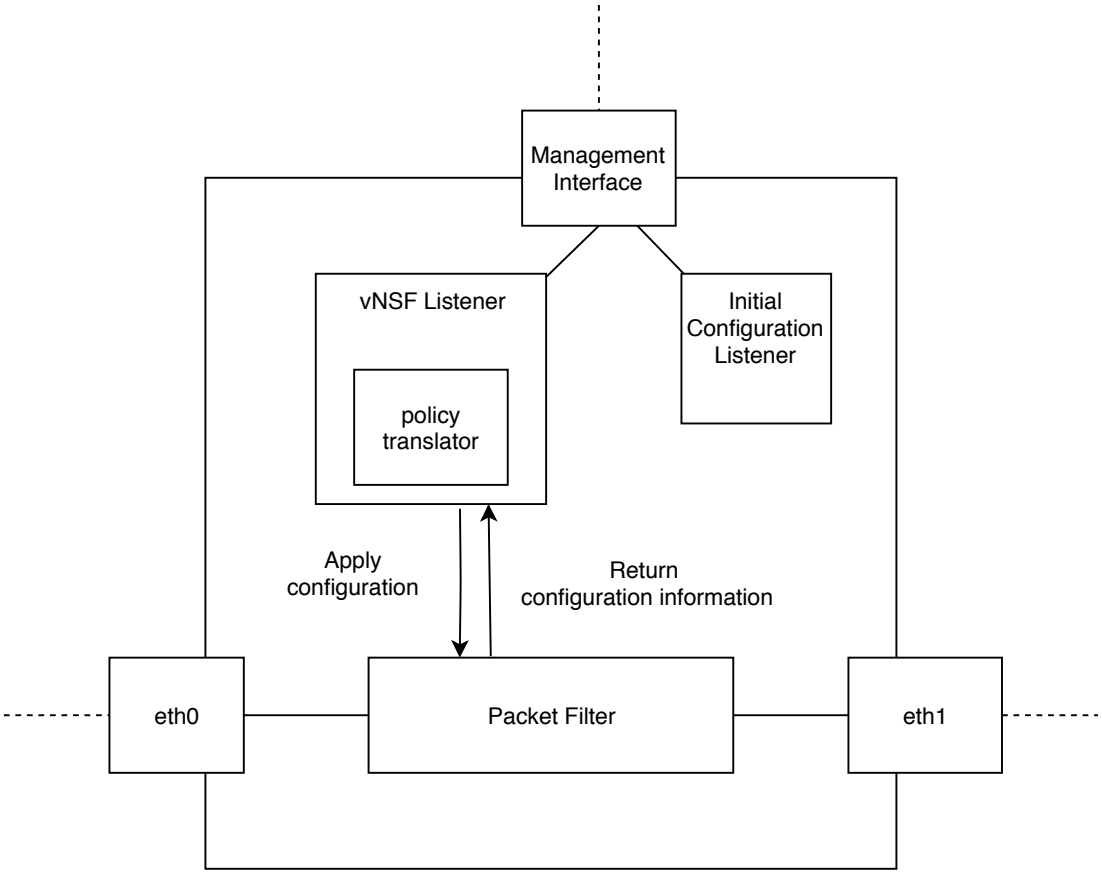

Figura 2.18. Architettura proposta per una vNSF in SHIELD

2.18 possono essere individuati alcuni blocchi principali:

- **Packet Filter:** rappresenta il modulo che fornisce le funzionalità di *packet filtering*. Gestisce il filtraggio dei pacchetti sulle interfacce connesse alla vNSF. In Fig. 2.18 vengono considerate per semplicità le sole interfacce `eth0` e `eth1`, ma possono essere presenti più di due interfacce da gestire. Nel caso specifico è rappresentato dalla tecnologia Netfilter/Iptables.
- **Initial Configuration Listener:** fornisce il modulo in ascolto per la configurazione iniziale della vNSF, effettuata dal vNFM.
- **vNSF Listener:** fornisce il modulo in ascolto per le richieste di configurazione ricevute dalla vNSF. Al suo interno presenta un modulo di traduzione delle policy, per passare dalla specifica MSPL alla configurazione effettiva del packet filter. Inoltre questo modulo è in grado di restituire informazioni sulla configurazione corrente del packet filter.

La soluzione illustrata presenta il punto di partenza del lavoro di tesi, stabilendo delle linee guida da seguire per l'implementazione della vNSF da realizzare. La soluzione illustrata, come anticipato, utilizza la tecnologia Netfilter/Iptables. Si vuole dunque realizzare una soluzione basata sulla tecnologia PF per ottenere, in accordo con quanto dimostrato nel test in sezione 2.2.3, una vNSF in grado di fornire prestazioni di rete migliori rispetto alla soluzione già realizzata.

Capitolo 3

Architettura

Questo capitolo si incentra sulle tecnologie utilizzate per poter realizzare un ambiente in cui la vNSF possa essere eseguita, oltre ad evidenziare come le tecnologie usate per la realizzazione di tale architettura interagiscono tra loro per istanziare e configurare la vNSF sviluppata.

3.1 Open Source MANO

Open Source MANO (OSM) [28] è uno strumento sviluppato da ETSI con lo scopo di fornire una soluzione open source di gestione ed orchestrazione in ambiente NFV. Questo strumento è utilizzato dunque per eseguire il ruolo di NFV MANO introdotto per il modello di funzionamento ETSI NFV descritto nella sezione 2.1.1. Nel contesto della tesi viene utilizzata la release FIVE di OSM. Il funzionamento di OSM può essere riassunto nella Fig. 3.1, in cui si evince che:

- OSM riceve le richieste attraverso un API accessibile da interfaccia grafica.
- OSM comunica con il *Virtual Infrastructure Manager* (VIM), ossia il componente atto alla gestione dell'NFVI, per istanziare le VNF ed i *Virtual Link* (VL) di connessione.
- OSM comunica con le VNF istanziate per effettuare tre fasi principali di configurazione chiamate day-0, day-1 e day-2, oltre a poter eseguire operazione a richiesta sulle VNF. Verranno fornite successivamente ulteriori informazioni riguardo le fasi citate.

Per permettere il corretto funzionamento di OSM è necessario che:

- Il VIM presenti un endpoint su cui viene esposta una API raggiungibile ed utilizzabile da OSM.
- Il VIM abbia una rete di *management*, ossia una rete che fornisca un indirizzo IP alle VNF istanziate.
- La rete di *management* sia raggiungibile da OSM.

3.1.1 Architettura OSM

In questa sezione vengono introdotti i moduli per poter fornire le funzionalità di gestione e come essi siano implementati in OSM. Per poter realizzare le funzionalità richieste al modulo MANO, OSM deve inglobare i moduli mostrati in verde nella Fig. 3.2 implementando funzionalità di:

- Gestione ed orchestrazione delle risorse messe a disposizione da una NFVI.
- Gestione del *lifecycle* di un NS.

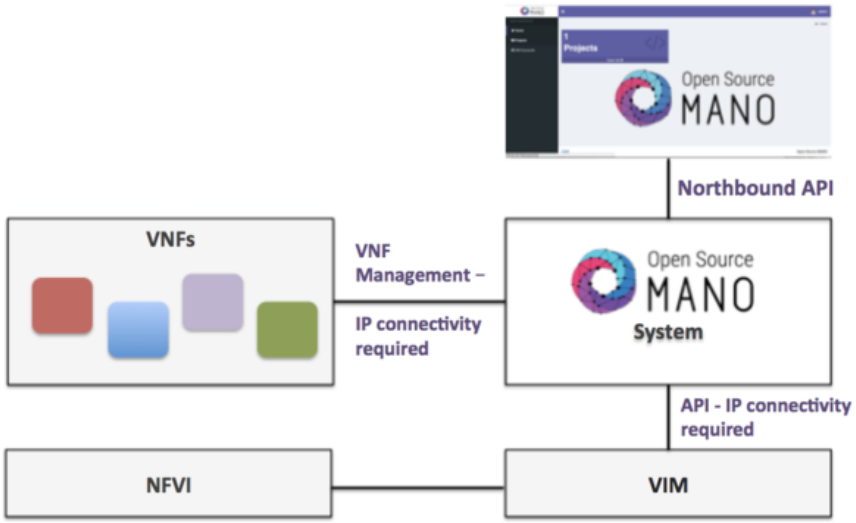

Figura 3.1. Funzionamento di Open Source MANO

- Gestione del ciclo di vita di una NFV.

Le prime funzionalità elencate sono tipiche dell'NFVO e vengono realizzate dai moduli *Service Orchestrator* (SO) e *Resource Orchestrator* (RO). In particolare il modulo SO riceve le richieste di gestione dei servizi contenuti all'interno di un database e le smista al modulo RO che si occupa di allocare le risorse necessarie all'istanziazione del NS richiesto. Inoltre l'SO è anche responsabile della comunicazione con moduli di alto livello come OSS/BSS. Il modulo RO, interfacciandosi con il VIM, si occupa dell'allocazione delle risorse fisiche dell'NFVI su nodi che prendono il nome di NFVI *Point of Presence* (PoP) mantenendo le informazioni sullo stato delle risorse istanziate. La funzionalità di gestione delle VNF viene invece realizzata dal modulo NFVM, il quale deve essere in grado di effettuare operazioni di configurazione e gestione sulle funzionalità di rete.

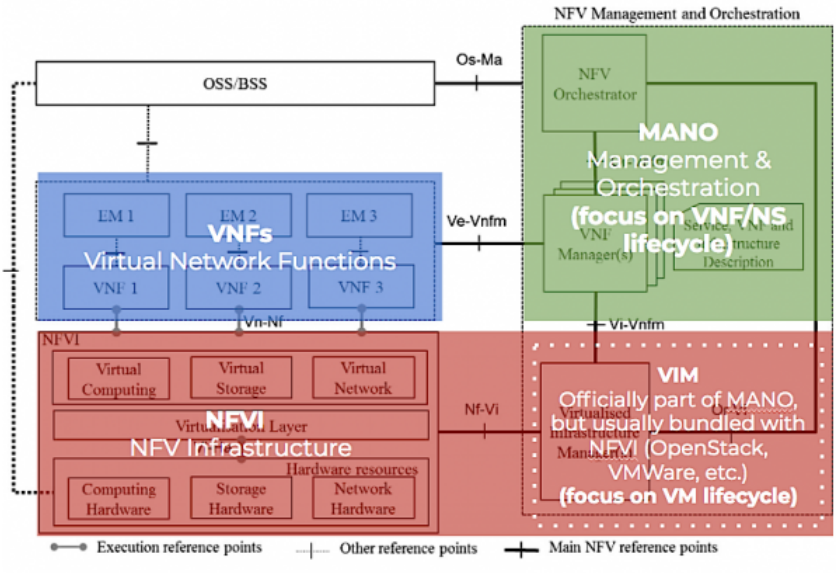

Figura 3.2. Divisione dei moduli dell'infrastruttura NFV

In OSM, in particolare nella release FIVE, i moduli implementati per realizzare tali funzionalità

sono presentati nella Fig. 3.3.

Il modulo RO utilizzato da OSM è OpenMANO. Esso fornisce, in modo open source, un insieme di moduli sviluppati in Python che supportano l'utilizzo di molteplici tipi di VIM ed orchestrano le risorse di diverse soluzioni presenti sul mercato. Il modulo RO riceve le richieste dal componente *Lifecycle Management* (LCM) e gestisce gli scenari e le istanze, le quali modellano rispettivamente i NS e le VNF. La funzionalità di gestione delle VNF viene invece implementata nel componente *VNF Configuration & Abstraction* (VCA). Esso ha come compito principale la gestione delle VNF tramite l'utilizzo di un unico VNF Manager che vada a far convergere la configurazione delle diverse VNF fornite in un unico applicativo. Il VNF Manager utilizzato da OSM viene individuato in Juju (vedi sez 3.3), per il quale fino alla release THREE di OSM venivano supportati particolari tipi di charm (proxy charm), mentre nell'architettura attuale vengono supportati anche i charm nativi grazie al modulo N2VC. Infine, il modulo SO, il quale si interfaccia con la GUI messa a disposizione da OSM e riceve le richieste dal client, è stato modellato come componente unico fino alla release THREE. Oltre ad interfacciarsi con la GUI, tra i compiti principali dell'SO rientrano la gestione del catalogo di OSM, l'onboarding delle VNF e dei NS e l'iterazione con il VNFM (Juju) per la configurazione delle VNF. La modellazione in un componente unico rendeva però difficile l'integrazione di nuovi moduli, in quanto l'SO rappresentava il punto di passaggio di tutte le comunicazioni tra le richieste ricevute da OSM ed i moduli dell'architettura. Ciò implicava dunque l'aggiunta di una parte di gestione delle nuove richieste all'interno di SO. A partire dalla release FOUR si è dunque deciso di dividere il modulo SO in due parti, individuate dalla Northbound Interface e dall'LCM. In questo modo vengono conservate le funzionalità descritte all'interno dell'LCM, introducendo però un'interfaccia per la ricezione delle richieste che permette di comunicare con i moduli all'interno dell'architettura di OSM, senza passare dall'LCM, inoltrando i messaggi su un bus.

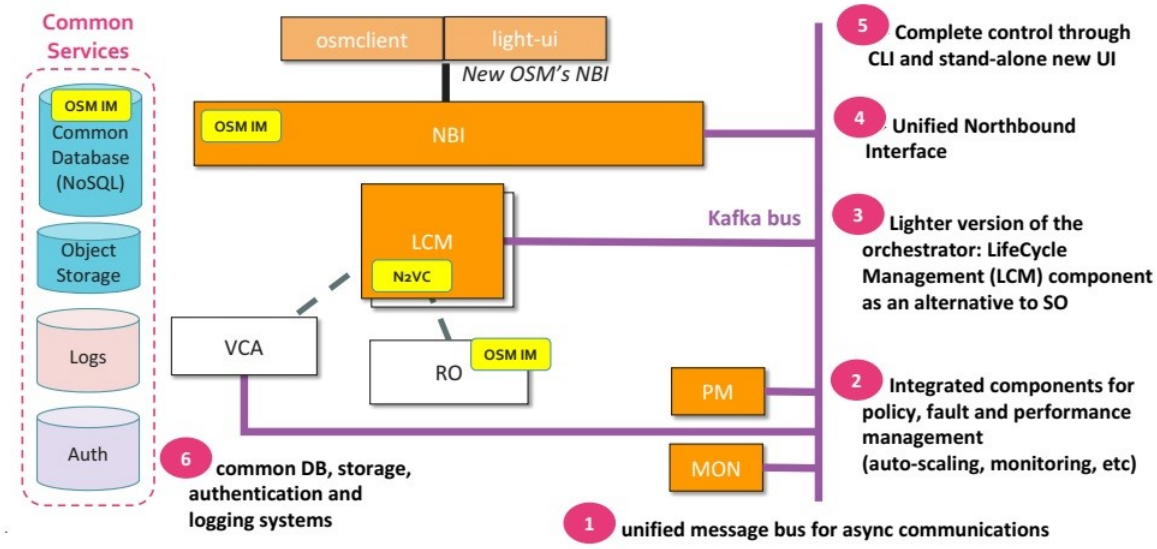

Figura 3.3. Architettura di OSM

OSM utilizza un bus di messaggistica unico, implementato con la tecnologia Apache Kafka [30], che permette lo scambio di messaggi asincrono tra i componenti che fanno parte dell'architettura. L'utilizzo di un bus unico permette, insieme allo sdoppiamento dell'SO, di collegare facilmente ad OSM dei moduli extra. I moduli che compongono l'architettura di OSM vengono implementati utilizzando il concetto di stack Docker, inteso come un insieme di servizi eseguiti su container Docker correlati tra loro con dipendenze comuni. Ogni modulo viene dunque modellato come un insieme di microservizi, ad eccezione del modulo VCA che è invece implementato utilizzando la tecnologia LXD ed è modellato per contenere il solo controllore Juju. Secondo quanto introdotto, per poter aggiungere un nuovo modulo all'interno dell'architettura bisogna implementare un nuovo stack Docker e collegarlo direttamente al bus di messaggistica unico all'interno di OSM. Infine sono presenti in OSM il *Policy Module* (PM) ed il *Monitoring Module* (MON), i quali sono responsabili

della gestione di policy, di guasti o di raccolta di informazioni sulle performance, oltre ad una parte di servizi comuni all'architettura, raggiungibili direttamente tramite la NBI.

3.1.2 Information Model

Il paradigma NFV, per permettere una descrizione dei componenti che cooperano alla fornitura di un servizio, mette a disposizione il concetto di descrittori. In particolare è possibile fornire le informazioni necessarie ad istanziare e gestire le funzioni virtuali disponibili tramite l'utilizzo di:

- *VNF Descriptor* (VNFD);
- *Virtual Link Descriptor* (VLD);
- *VNF Forwarding Graph Descriptor* (VNFFGD);
- *Physical Network Function Descriptor* (PNFD);
- *NS Descriptor* (NSD).

Tali descrittori devono tener conto della grande eterogeneità di tecnologie presenti in grado di fornire soluzioni di orchestrazione e gestione delle VNF.

OSM propone un ampio IM utilizzato per permettere l'interoperabilità tra differenti servizi di NFVI e diversi sistemi OSS. Tale IM supporta l'utilizzo dei NSD per la descrizione dei servizi di rete e dei VNFD per la descrizione delle funzioni che compongono i NS. Il modello è molto ricco e viene fornita una breve descrizione dei descrittori, per una conoscenza approfondita è possibile fare riferimento alla guida OSM [29].

Il NSD contiene la descrizione del NS inteso come gruppo di VNF con una specifica topologia utilizzate per la fruizione del servizio. Esso permette infatti di definire i *Virtual Link* (VL) utilizzati per il collegamento delle VNF, in che modo le VNF debbano essere attraversate per la fruizione del servizio ed i punti di connessione del NS, in modo da permettere un'eventuale catena di servizi. Inoltre è possibile definire operazioni di configurazione comuni all'intero gruppo di VNF che costituiscono il NS.

Il VNFD fornisce invece una descrizione dettagliata della singola VNF che può essere utilizzata in un NS. Tra gli elementi più importanti che è possibile specificare in un VNFD si trovano il *Virtual Deplyment Unit* (VDU), inteso come unità virtuale effettivamente istanziata, ed l'*Internal Virtual Link* (IVL), che modella il collegamento tra le unità virtuali che compongono una VNF. Per la VDU vengono definite principalmente informazioni relative a:

- Immagine da utilizzare per l'istanza virtuale.
- Interfacce della VNF, includendo informazioni sull'interfaccia di management.
- Definizione del *flavor*, inteso come risorse in termini di CPU, memoria e spazio di archiviazione da utilizzare per la VNF.

Inoltre in questo descrittore vengono definite le operazioni di configurazione da effettuare sulla VNF. In questo caso è possibile specificare operazioni da effettuare appena terminata la fase di istanziazione, in modo da ottenere una corretta configurazione iniziale. Vengono inoltre definite operazioni eseguibili su richiesta, solitamente utilizzando il controllore Juju, in modo da ottenere una gestione della VNF durante il *lifecycle* grazie all'utilizzo di opportuni parametri.

Come mostrato in Fig. 3.4, l'architettura OSM tramite l'utilizzo dell'IM incapsula le operazioni da effettuare ad un livello più alto, in modo da rendere la configurazione indipendente dalle tecnologie utilizzate a livello di infrastruttura.

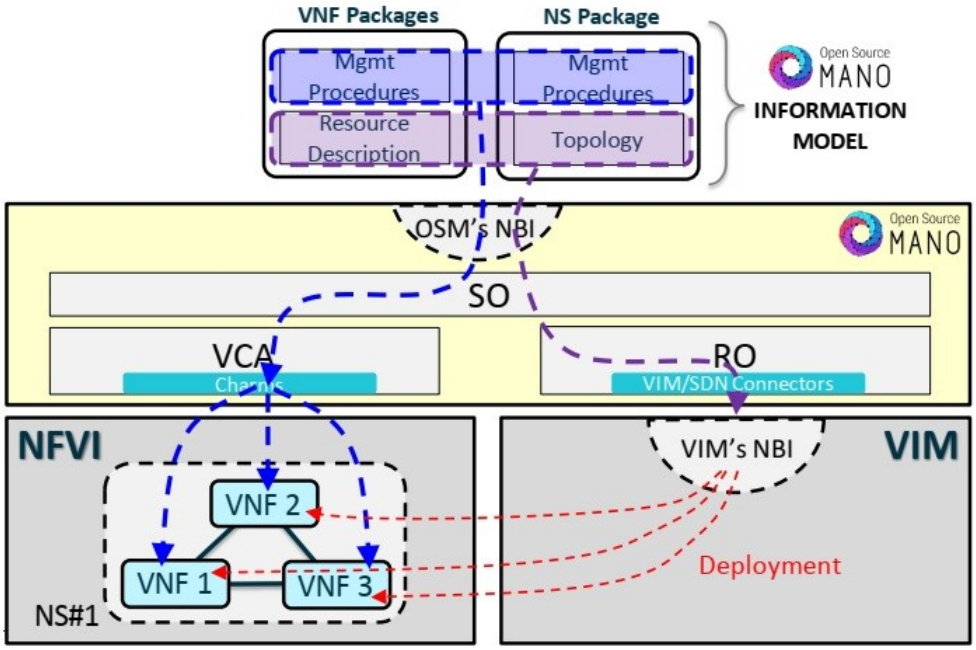

Figura 3.4. Information Model nell'architettura OSM

3.1.3 Step di configurazione della VNF

Come introdotto, OSM mette a disposizione tre step di configurazione:

- **Day-0:** questo tipo di configurazione avviene nel momento in cui la VNF viene istanziata.
- **Day-1:** questo tipo di configurazione avviene dopo che la VNF è stata istanziata, oltre a poter eseguire operazioni di configurazione al verificarsi di specifici eventi.
- **Day-2:** questo tipo di configurazione permette di effettuare operazioni a livello di network service, effettuando operazione che vadano a modificare la configurazione dell'intero servizio.

Per la configurazione day-0 uno strumento largamente utilizzato è rappresentato da *cloud-init* [31]. Esso permette di effettuare operazioni di configurazione su immagini di sistemi operativi utilizzati nell'ambito cloud. Ciò permette di partire da immagini base, per poi poter effettuare le operazioni di configurazione necessarie per le esigenze richieste. L'utilizzo di cloud-init permette ad esempio di specificare:

- dimensioni dell'istanza utilizzata per la VNF, in termini di *flavor*;
- utenti e gruppi dell'istanza;
- dipendenze da installare;
- parametri di configurazione dei servizi di rete;

tale strumento non risulta però direttamente compatibile con FreeBSD, cioè il sistema operativo utilizzato nel contesto della tesi per la realizzazione del packet filter. A tale scopo viene utilizzato bsd-cloudinit [32]. Questa tecnologia nasce come adattamento di cloud-init per il sistema operativo FreeBSD, in modo da poter ottenere alcuni dei vantaggi descritti. In particolare, nel contesto della tesi, tale tecnologia viene utilizzata per ottenere istanze di FreeBSD su OpenStack per le quali sia possibile specificare il *flavor*. Tra le funzionalità non supportate da bsd-cloudinit rientra la possibilità di specificare utenti nel momento in cui una nuova istanza viene creata. È dunque

risultato necessario pre-caricare un utente nel momento in cui l'immagine per OpenStack è stata creata, tale procedimento viene meglio descritto nel manuale dello sviluppatore (sez. B).

Lo step di configurazione nel quale viene effettuato il maggior numero di operazioni nel contesto della tesi è il day-1. Questa fase viene realizzata utilizzando lo strumento Juju ed implementa la maggior parte delle operazioni inerenti la configurazione iniziale della vNSF, tramite l'esecuzione del playbook Ansible (vedi sez. 3.4). Inoltre tramite Juju vengono implementate le operazioni atte a configurare il packet filter durante il suo ciclo di vita.

3.2 OpenStack

OpenStack [33] è un progetto open source per il cloud computing che nasce dalla collaborazione tra NASA e Rackspace. Viene rilasciato la prima volta nel 2011 per il sistema operativo Ubuntu, in seguito viene rilasciato per altre distribuzioni con ulteriori release. La versione utilizzata nel contesto della tesi è OpenStack *Pike*. Esso fornisce un servizio cloud di tipo IaaS, in quanto mette a disposizione risorse virtualizzate all'utilizzatore (e.g. server, risorse di rete, risorse di storage) tramite l'utilizzo di diversi componenti. OpenStack viene utilizzato in quanto riesce a fornire la parte di infrastruttura (NFVI) ed il modulo VIM per il funzionamento di OSM mostato in Fig. 3.1.

3.2.1 Architettura di OpenStack

OpenStack è organizzato in più moduli, indipendenti tra loro, che prendono il nome di OpenStack *service*. I singoli servizi interagiscono tra loro utilizzando delle API pubbliche, secondo lo schema mostrato in Fig. 3.5. Essi sono composti internamente da diversi processi, tra cui un servizio in ascolto per le richieste fornite tramite l'utilizzo dell'API. Tali richieste vengono pre-processate e smistate agli altri processi del servizio, che provvederanno a soddisfarle. Tra gli OpenStack *service*, si possono distinguere alcuni servizi di base per il funzionamento di OpenStack, individuati da Nova, Neutron, Cinder, Keystone, Glance e Horizon.

Vi sono altri servizi che forniscono funzionalità aggiuntive, ma non indispensabili per una installazione minima di Openstack, come Heat, Celiometer, Trove, Sahara, Magnum, Ironic e Tracker.

3.2.2 Servizi di OpenStack

In questa sezione si andrà a fornire una panoramica sui servizi di base messi a disposizione da OpenStack.

Nova

Nova fornisce il servizio di Compute. Esso è responsabile di eseguire la parte di hypervisor, ossia effettua la gestione delle istanze virtuali. Di base il servizio di Compute utilizza KVM, ma Nova fornisce supporto anche per altre tecnologie di virtualizzazione (e.g. VMware, Xen, Hyper-V), oltre a fornire un supporto di base ai container. Sul nodo di Compute viene inoltre eseguito un agente del servizio di Networking che è responsabile del collegamento delle istanze alle reti virtuali, garantendo un servizio di *firewalling* tramite l'utilizzo di gruppi di sicurezza. Nova è composto da processi differenti che cooperano tra loro per fornire tale servizio. Le richieste dei client vengono ricevute tramite una API REST, mentre i componenti comunicano internamente attraverso un meccanismo di messaggi RPC. Tali messaggi vengono scambiati tramite l'utilizzo di una libreria, chiamata **oslo.messaging**, che fornisce un'astrazione della coda di messaggi. I componenti principali mostrati in Fig. 3.6, i quali permettono il funzionamento di Nova sono:

- DB: database SQL condiviso tra i componenti per lo storage dei dati.

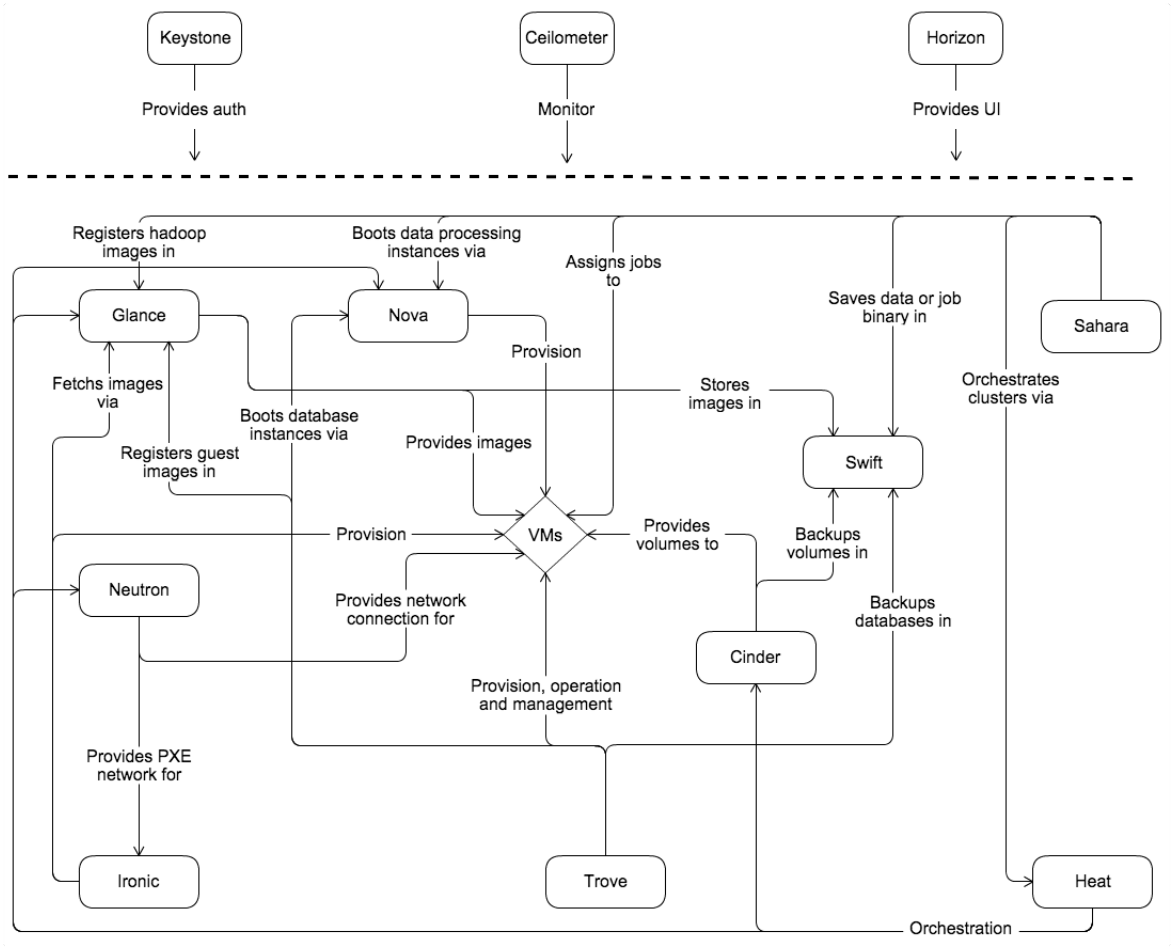

Figura 3.5. Architettura concettuale di OpenStack

- API: componente che riceve le richieste HTTP, le elabora e le smista ai componenti interni tramite l'utilizzo delle code e del protocollo HTTP.
- Scheduler: decide su quali host indirizzare le istanze.
- Compute: gestisce la comunicazione tra l'hypervisor e le VM.
- Conductor: gestisce le richieste che richiedono coordinamento e agisce come proxy per il database.
- Placement: tiene traccia delle risorse messe a disposizione e del loro utilizzo.

Neutron

Neutron è un servizio che ha come scopo quello di fornire il *Networking as a service* (NaaS). Questo servizio di Networking fornisce l'astrazione di reti, sottoreti e router, permettendo di creare topologie di rete complesse ed in grado di supportare diversi tipi di servizi (e.g. firewall, load balancer, VPN). Ogni configurazione di Networking presenta una rete esterna. Essa non è soltanto una rappresentazione virtuale di una rete, ma presenta un vero e proprio collegamento ad una interfaccia fisica del dispositivo su cui essa risiede. Le VM presenti su questa rete sono in grado di comunicare con il mondo esterno ad OpenStack e qualunque indirizzo IP esposto su tale rete risulta raggiungibile dall'esterno. Oltre al collegamento con la rete esterna, ogni configurazione creata da Neutron presenta anche una rete interna. Su questo tipo di rete avviene la comunicazione tra le VM, che risulteranno raggiungibili direttamente solo dalle VM connesse

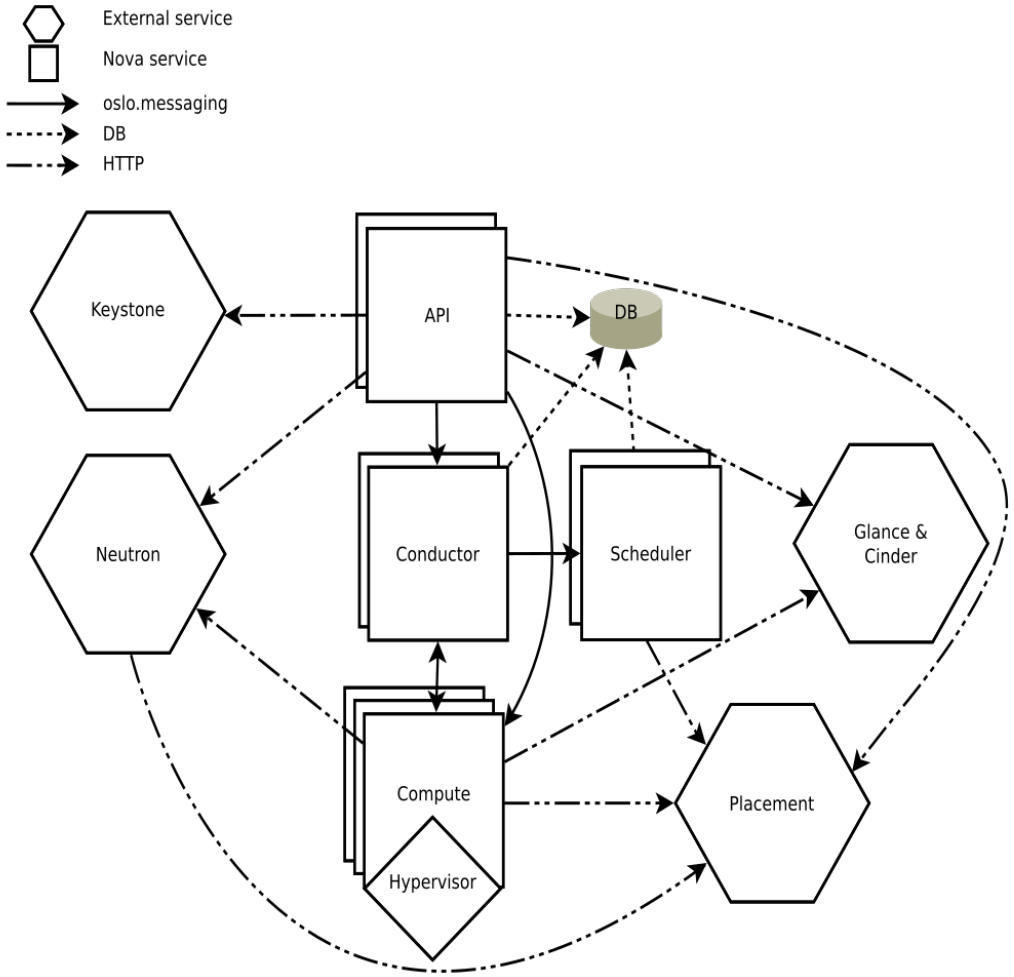

Figura 3.6. Componenti principali di Nova

alla medesima rete. Le configurazioni definite tramite il servizio di Networking supportano anche il concetto di *security groups*. Questi gruppi permetto di definire delle regole di accesso alle VM, suddividendo tali politiche in gruppi di applicazione. In questo modo si possono filtrare o rendere accessibili determinati servizi della VM soltanto ad una certa tipologia di utilizzatori. I componenti del servizio di Networking, dei quali è possibile avere una visione architetturale in Fig. 3.7, sono:

- **Neutron server:** questo servizio viene eseguito sul nodo di controllo ed è responsabile della gestione delle chiamate dell'API. Questo server coordina gli agent per fornire il servizio di rete richiesto tramite l'utilizzo della *message queue*. Richiede inoltre l'accesso ad un database persistente, realizzato da plugin che utilizzano il protocollo di scambio di messaggi *Advanced Message Queuing Protocol* (AMQP).

- **Plugin agent:** questo agente viene eseguito su ogni compute node per gestire la configurazione degli switch virtuali locali o *vswitch*. Tale agente richiede l'accesso alla *message queue*.

- **DHCP agent:** questo agente viene eseguito su ogni nodo di rete e fornisce il servizio di DHCP per quel nodo. Tale agente richiede l'accesso alla *message queue*.

- **L3 agent:** questo agente viene eseguito su ogni nodo di rete e fornisce alle VM connettività di livello 3 per la rete esterna. Tale agente richiede l'accesso alla *message queue*.

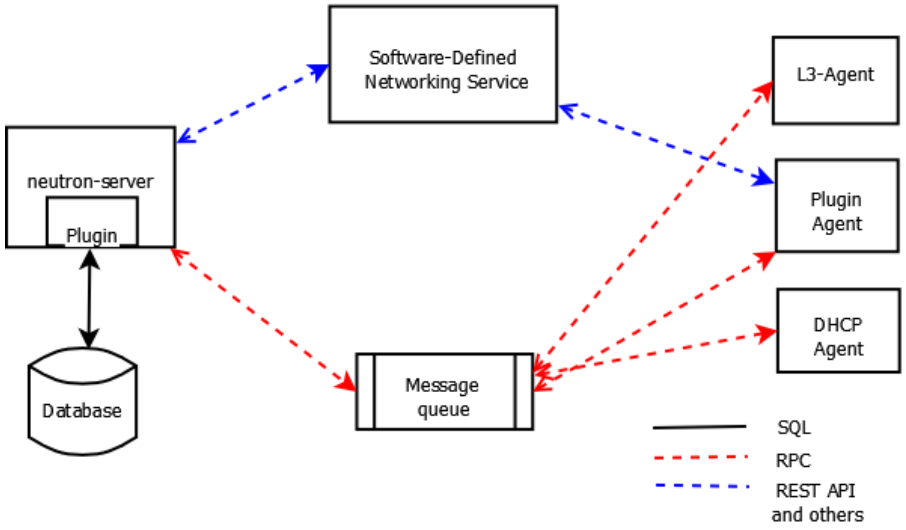

Figura 3.7. Componenti del servizio di Networking (Neutron)

- **Network provider service (SDN server/service):** fornisce dei servizi aggiuntivi che permetto la gestione delle reti presenti sui nodi tramite software. Questo servizio SDN può interagire con il neutron server, i plugin e gli agenti tramite l'utilizzo di una API REST.

Cinder

Cinder fornisce il servizio di Block Storage. Esso virtualizza la gestione di blocchi di storage messi a disposizione da OpenStack, fornendo una API che permette di utilizzare tali volumi senza tener conto dei dettagli sui dispositivi sui quali sono allocati. Tramite Cinder è possibile assegnare volumi persistenti alle VM, in quanto OpenStack, come comportamento di default, elimina lo storage assegnato ad una VM alla fine del ciclo di vita dell'istanza.

Keystone

Keystone fornisce un servizio che mette a disposizione una API per effettuare l'autenticazione del client e fornire le autorizzazioni necessarie all'utilizzo dei servizi messi a disposizione da OpenStack. Esso è organizzato su un gruppo di servizi interni, utilizzati in modo combinato tra di essi. Ad esempio un utente/progetto viene autenticato tramite il servizio interno *Identity*, ed in caso di autenticazione riuscita il servizio *Token* restituisce un token che fornisce al client l'autorizzazione ad utilizzare il servizio richiesto.

Glance

Glance fornisce il servizio di *Image*. Esso è responsabile della memorizzazione delle immagini software e dei metadati relativi ad una VM. Le richieste di salvataggio e di ritrovamento dei metadati o delle immagini relative ad una VM vengono effettuate tramite una API RESTful messa a disposizione da Glance. È possibile memorizzare le immagini utilizzate da Glance su un semplice filesystem o su sistemi appositi per la memorizzazione di oggetti (e.g. OpenStack Swift).

Horizon

Horizon fornisce una interfaccia grafica per OpenStack. Tramite tale interfaccia è possibile accedere e configurare i servizi di OpenStack. Horizon fornisce un insieme di astrazione delle API dei progetti principali, in modo da semplificare l'utilizzo di tali servizi agli utilizzatori.

3.3 Juju

Juju [34] è uno strumento open source per modellare il software operativo all'interno di cloud pubblici e privati. Juju permette di gestire, configurare e scalare le applicazioni web in modo efficiente, oltre a permettere le medesime operazioni su server fisici, su container o su piattaforme *IaaS* (e.g. Openstack, AWS). L'utilizzo di Juju risulta particolarmente utile nell'ambito dei moderni ambienti, in quanto raramente un'applicazione viene istanziata in modo isolato. Così come descritto per OpenStack, che presenta un'architettura in cui vengono inclusi diversi moduli che devono essere configurati e connessi tra di loro, così le applicazioni più semplici hanno spesso bisogno di altre applicazioni (e.g. database, web server) per poter funzionare in modo appropriato. Le operazioni di configurazione di una specifica applicazione, le dipendenze, e gli eventi significativi alla gestione del ciclo di vita dell'applicazione, vengono incapsulati nei *charm*, che permettono inoltre di condividere configurazioni di vari tipi di applicazione per permettere un riutilizzo del charm stesso. Juju può essere utilizzato con altri strumenti di gestione della configurazione, come *Ansible* (paragrafo 3.4), *Chef* o *Puppet*. Tali strumenti forniscono soluzioni per la scrittura di file di configurazione che possano risultare semplici da utilizzare e riadattare. Come detto Juju raccoglie la logica per il funzionamento delle applicazioni nel tempo nei *charm*, senza aver conto delle specifiche della macchina su cui tali operazioni debbano essere eseguite. Ciò permette di poter specificare tale logica utilizzando il linguaggio e gli strumenti che si preferiscono. Questo strumento viene introdotto in quanto viene utilizzato nel modulo VCA di OSM per la configurazione di una VNF.

3.3.1 Concetti principali

In questa sezione verranno introdotti i concetti principali utilizzati nel contesto di Juju.

Cloud

Un cloud, nel contesto di Juju, rappresenta una risorsa in grado di fornire istanze per le unità dell'applicazione da istanziare, e possibilmente capacità di storage. Tale risorsa può essere fornita da servizi di cloud pubblici (e.g. AWS, Microsoft Azure), privati (e.g. OpenStack) o macchine fisiche (e.g. MAAS, LXD).

Controller

Un *Juju controller* è la prima istanza creata da Juju su un cloud. Tale istanza fornisce un nodo di gestione centrale sul cloud controllato, che eseguirà tutte le operazioni richieste dal client Juju. Il controller fornisce a Juju l'accesso all'API del cloud.

Model

Un *model* descrive lo spazio associato ad un controller, in cui le unità applicative vengono distribuite. Un controller può avere un numero indefinito di *model* associati e un *model* può contenere al suo interno un numero indefinito di *Juju machine*. Il *controller model* contiene tipicamente una sola *Juju machine*, coincidente con il controller stesso. Vengono definiti *regular model* i modelli in cui sono distribuite tutte le altre unità dell'applicazione. Un esempio viene mostrato in Fig. 3.8.

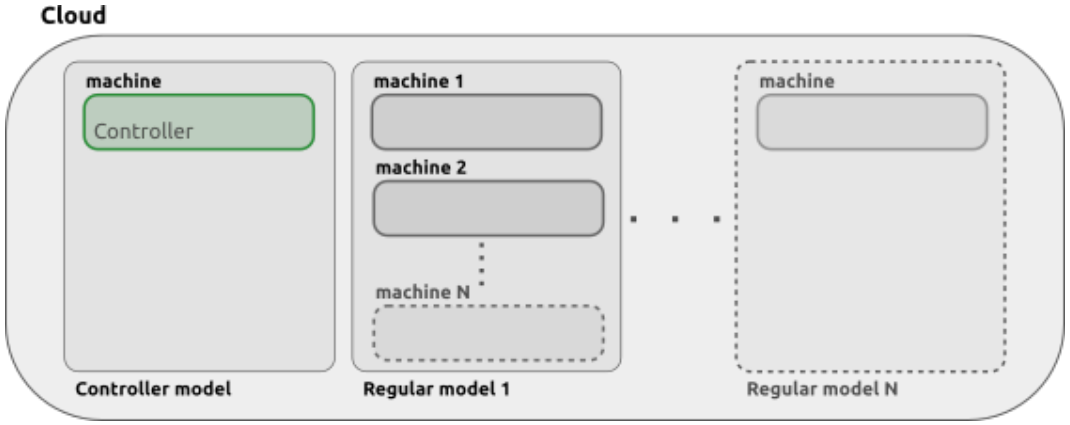

Figura 3.8. Esempio di *Juju model*

Charm

Un *Juju charm* rappresenta la risorsa in cui si raccolgono tutte le istruzioni necessaria all'avvio e alla configurazione di nuove unità applicative. Un *charm* può essere sviluppato ad-hoc per una specifica applicazione, oppure essere scelto dal *Charm Store* messo a disposizione da *Juju*. Lo scenario di utilizzo più semplice, mostrato in Fig. 3.9, prevede il deploy di un singolo *charm* senza l'utilizzo di ulteriori opzioni. Questo scenario prevede la creazione di una nuova istanza su una *Juju machine* all'interno del cloud. Su tale istanza verranno eseguite le istruzioni raccolte all'interno del *charm*, in modo che tale unità applicativa possa essere configurata correttamente. Un *charm* può essere definito come subordinato o principale. Il subordinato viene utilizzato

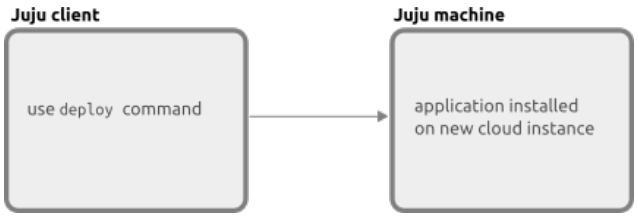

Figura 3.9. Scenario di utilizzo di un juju charm

per estendere le funzionalità di un charm principale, con la differenza che nessuna nuova unità viene creata per tale *charm*. Il principale prevede invece la creazione di una nuova unità, come descritto nello scenario di utilizzo di base. Maggiori dettagli sui *Juju charm* verranno mostrati nella sezione 3.3.2.

Container

Un *Juju container* è un termine generico che identifica macchine LXD-based o KVM-based.

Bundle

Un *Juju bundle* rappresenta una raccolta di *charm*. Tali *charm* vengono configurati e collezionati in un singolo *bundle* per automatizzare deploy e configurazione di soluzioni *multi-charm*. Ad esempio nel caso di una applicazione che utilizza un database ed un web service, il bundle può contenere i charm utilizzati per la configurazione dei due componenti dell'applicazione e la relazione tra essi.

Machine

Una *Juju machine* identifica una istanza cloud richiesta da Juju. Spesso una machine ospita una singola unità di una applicazione distribuita, come mostrato in Fig. 3.10. Risulta però possibile ospitare un numero arbitrario di unità sulla singola Juju machine.

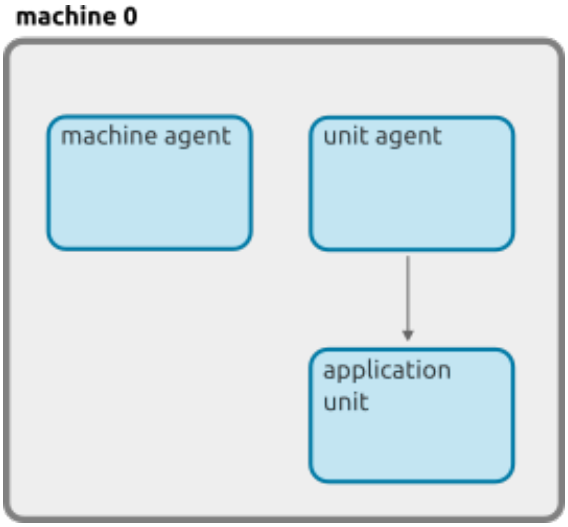

Figura 3.10. Juju machine con singola unità applicativa

Unit e Application

Una *Juju unit* rappresenta un software distribuito che viene utilizzato per fornire una applicazione. Un'applicazione semplice può essere formata da una singola unità applicativa, ma è anche possibile avere applicazioni formate da più unità in esecuzione su *machine* multiple. Ogni unità dell'applicazione avrà lo stesso charm, le stesse *relations* e la stessa configurazione.

Leader

Con il termine *leader* si intende l'unità dell'applicazione che è responsabile di fornire lo stato e la configurazione dell'applicazione. In ogni momento del ciclo di vita di una applicazione, essa può avere una singola unità leader. Le singole unità possono acquisire il ruolo di leader e mantenerlo per il tempo necessario.

Endpoint

Un *endpoint* fornisce un punto di connessione tra applicazioni, in modo da poter stabilire una relazione tra di esse. Gli *endpoint* possono ricoprire tre ruoli differenti:

- *requires*: L'endpoint può utilizzare i servizi messi a disposizione da un altro charm sull'interfaccia fornita.
- *provides*: L'endpoint rappresenta un servizio che può essere utilizzato dall'endpoint di un altro charm.
- *peers*: L'endpoint può coesistere con l'endpoint di un altro charm stabilendo una relazione peer-to-peer.

Gli endpoint vengono specificati nel file ***metadata.yaml*** specificando il nome, il ruolo e l'interfaccia. Ad esempio nel file *metadata.yaml* del charm di 'wordpress' vengono descritti i seguenti endpoint:

```
requires:
  db:
    interface: mysql
  nfs:
    interface: mount
  cache:
    interface: memcache
provides:
  website:
    interface: http
peers:
  loadbalancer:
    interface: reversenginx
```

Listing 3.1. Esempio di endpoint nel file metadata.yaml

Interface

Una *interface* rappresenta il protocollo di comunicazione utilizzato in una *relation* tra le applicazioni.

Relation

Una *relation* è una connessione tra due applicazioni. Essa avviene connettendo due endpoint di due diverse applicazioni, a patto che tali endpoint siano compatibile per interfaccia e ruolo. È possibile connettere:

- Un endpoint con ruolo *requires* ad un endpoint con ruolo *provides*.
- Un endpoint con ruolo *provides* ad un endpoint con ruolo *requires*.
- Un endpoint con ruolo *peer* ad un endpoint con ruolo *peer*.

La Fig. 3.11 mostra delle possibili connessioni per l'esempio 3.1, in cui vengono definiti alcuni endpoint utilizzati dal charm di "wordpress". In questo esempio vengono connesse le interfacce

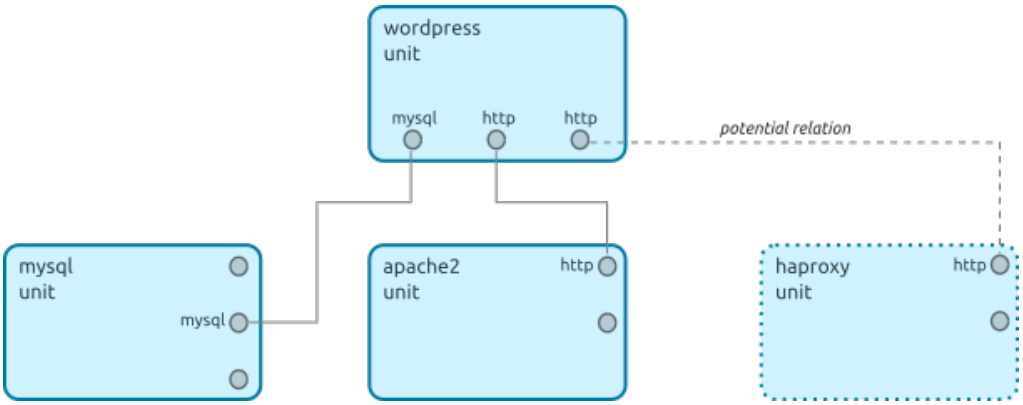

Figura 3.11. Esempi di *relation*

http, che fornisce un servizio, e *mysql*, che richiede un database. In modo simile si può connettere qualunque altro charm che supporta tali interfacce.

Client

Un *Juju client* rappresenta il software *command line interface* (CLI) utilizzato per gestire Juju. Tale software si connette ai *Juju controllers* e viene utilizzato per gestire ed inizializzare nuove unità applicative sul cloud.

Agent

Un *Juju agent* è un software in esecuzione su ogni *Juju machine*. Un *Juju agent* può essere di due tipi:

- *machine agent*: opera a livello macchina.
- *unit agent*: opera a livello di unità applicativa.

Ogni *Juju machine* di tipo non-controller ha almeno due *agent* in esecuzione, uno di tipo machine ed uno di tipo unit. Una *Juju machine* di tipo controller presenta generalmente un singolo *machine agent* in esecuzione. I *machine agent* creano e gestiscono i propri *unit agent* che vengono richiesti sulla *Juju machine* a cui essi appartengono. Gli *unit agent* sono responsabili dell'esecuzione delle attività richieste dal *Juju charm*. Tutti gli *agent* tengono traccia dello stato e dei suoi cambiamenti, rispondendo alle azioni che tale stato comporta e restituendo informazioni aggiornate al controller.

3.3.2 Juju charm

Il *Juju charm* rappresenta un blocco essenziale al funzionamento di Juju, poichè come anticipato esso raccoglie le informazioni e le operazioni necessarie all'istanziazione ed alla configurazione di una nuova unità. Essi possono essere sviluppati come soluzione unica, oppure incorporati in *Juju bundle* per fornire soluzioni utilizzabili per configurazioni complesse. Un *juju charm* può essere specificato utilizzando svariati linguaggi di programmazione, quali Ruby, PHP, Python o semplicemente script bash. Inoltre, come anticipato, un charm è in grado di utilizzare tecnologie di gestione della configurazione come *Ansible*, il quale viene integrato con il charm realizzato nel contesto del lavoro di tesi.

Un *Juju charm* presenta un'organizzazione ben definita. In particolare esso deve presentare un singolo file denominato **metadata.yaml** in cui vengono raccolte informazioni relative al charm. Un charm che presenta il solo file metadata.yaml viene considerato come valido, pur non contendendo nessuna funzionalità. Oltre al file in cui sono raccolte le informazioni del charm, possono essere presenti elementi opzionali, raggruppati nel seguente modo:

- **Directory hooks:** cartella contene eseguibili invocati da Juju in diversi istanti di tempo per la gestione dell'applicazione. Un charm deve implementare almeno un *hook* per essere in grado di eseguire una effettiva funzionalità.
- **Directory actions:** contiene i file eseguibili, nominati in modo coerente con l'azione da eseguire, che l'utente può invocare tramite Juju quando desiderato.
- **Directory tests:** contiene gli script di test, eseguiti per essere sicuri del corretto funzionamento del servizio messo a disposizione dal charm.
- **Actions.yaml:** definisce le azioni che il charm è in grado di fornire. Deve essere presente se viene utilizzata la directory actions.
- **Config.yaml:** definisce le operazioni di configurazione del charm.
- **Icon.svg:** utilizzato come icona da visualizzare all'interno della GUI di Juju o all'interno del charm store.
- **README:** file di supporto all'utilizzo del charm reso disponibile agli utenti del charm store.

Risulta inoltre interessante evidenziare le caratteristiche che portano alle definizioni di *reactive charm* e *layered charm*. I *reactive charm* rispondono al paradigma di *reactive programming*, ossia esecuzione di codice in risposta a determinati eventi. Juju mette a disposizione il package **charms.reactive** che sfrutta la logica booleana per stabilire quando le condizioni per eseguire

determinate azioni sono soddisfatte. Gli eseguibili che rispondono a tale paradigma vengono salvati nella directory **reactive** del charm. Il concetto di *layered charm* sfrutta il concetto di *Layer* in Juju. È infatti possibile strutturare un charm su livelli differenti, dando la possibilità di riutilizzare parti comuni a più charm semplicemente importando il livello che realizza la funzionalità desiderata. Esistono tre diverse tipologie di *Layer*:

- **Base/Runtime:** questo livello raccoglie le funzionalità di base che un charm esegue, come ad esempio la gestione delle dipendenze o di operazioni di preparazione all'esecuzione di script.
- **Interface:** questo livello è responsabile della comunicazione che viene effettuata su una *relation* tra due applicazioni. In questo livello viene incapsulato un protocollo di interfaccia e deve essere realizzato in Python.
- **Charm layer:** questo livello contiene la logica applicativa del charm realizzato e si basa sui livelli precedenti.

Per poter sfruttare il concetto di *layered charm* bisogna includere il file **layer.yaml** in cui vengono dichiarati i livelli utilizzati. La struttura di un charm che sfrutta i due paradigmi descritti viene mostrata in Fig. 3.12.

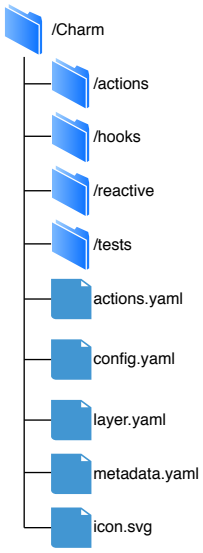

Figura 3.12. Organizzazione strutturale di un Juju Charm

3.4 Ansible

Ansible [35] è uno strumento di automazione che permette la configurazione, anche complessa, di sistemi IT, che si pone come obiettivo principale la semplicità nell'utilizzo, oltre a concentrarsi sugli aspetti di sicurezza e affidabilità. Per poter raggiungere l'obiettivo della semplicità, Ansible è stato progettato per utilizzare un linguaggio che fosse altamente leggibile e comprensibile dagli umani. Inoltre, utilizzando come strumento di trasporto principale OpenSSH (mettendo comunque a disposizione varie alternative), Ansible non richiede l'installazione di alcun agente sulla macchina da configurare, sollevando l'utilizzatore da qualsiasi responsabilità riguardante installazione o aggiornamento di daemon remoti. Per quanto riguarda gli aspetti di sicurezza, oltre ad utilizzare strumenti di trasporto ritenuti sicuri, Ansible utilizza le credenziali del sistema operativo per accedere alle macchine remote da configurare, permettendo inoltre una facile integrazione con sistemi centralizzati di gestione dell'autenticazione, quali Kerberos o LDAP. Per poter avere una visione migliore del funzionamento di Ansible, vengono introdotti i seguenti concetti di base:

- **Nodo di controllo:** qualunque macchina su cui è installato Ansible, e da cui risulta possibile eseguire comandi e playbook. Qualunque macchina su cui sia installato Python può fungere da nodo di controllo, a meno che non sia una macchina Windows. Questo ruolo può essere ricoperto da più nodi contemporaneamente.
- **Nodi gestiti:** nodi di rete che vengono gestiti tramite Ansible. Questi nodi vengono anche chiamati "host" e su di essi non è necessario avere un'installazione di Ansible.
- **Inventario:** lista di nodi gestiti, in cui vengono comunemente salvate informazioni riguardanti i singoli "host" (e.g. Indirizzo IP, porta SSH). Vengono anche chiamati "hostfile".
- **Modulo:** l'unità di codice eseguita da Ansible. Si può invocare un singolo modulo in un task, o invocare molteplici moduli in un playbook.
- **Task:** rappresenta la singola azione di configurazione nel contesto di Ansible.
- **Playbook:** lista che raggruppa diversi task, in modo da poterne eseguire alcuni gruppi nel momento in cui essi sono necessari.

3.4.1 Inventario

Ansible sfrutta il concetto di inventario per poter lavorare su host molteplici all'interno dell'infrastruttura da gestire. Le informazioni relative ad un'inventario vengono salvate in file che possono essere scritti in diversi formati (e.g. YAML, INI). La locazione predefinita per il salvataggio delle informazioni è /etc/ansible/hosts, ma è possibile salvare le informazioni in diversi file ed anche utilizzarne molteplici in contemporanea. Nel formato INI, un esempio di inventario è:

```
[webservers]
one.example.com
two.example.com

[dbservers]
three.example.com
four.example.com
five.example.com
```

che ha il suo equivalente nel formato YAML:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        one.example.com:
        two.example.com:
    dbservers:
      hosts:
        three.example.com:
        four.example.com:
        five.example.com:
```

Nell'esempio mostrato, i diversi host vengono gestiti in due gruppi, ossia *webservers* e *dbservers*, effettuando un raggruppamento degli host in base a quale servizio essi forniscono. All'interno dell'inventario è possibile specificare lo stesso host in gruppi multipli, oltre ad avere la possibilità di avere gruppi nidificati. In questo modo è possibile organizzare l'hostfile in modo da avere raggruppamenti in base a:

- **Cosa:** raggruppamento in base al servizio che viene fornito.

- **Dove:** raggruppamento dei servizio in base alla loro posizione geografica.
- **Quando:** raggruppamento in base alle operazioni da eseguire nei vari livelli del lifecycle.

Ansible mette a disposizione due gruppi predefiniti, **all** e **ungrouped**. Al primo appartengono tutti gli host definiti nell'inventario, nel secondo rientrano gli host che appartengono al solo gruppo all e in nessun altro gruppo. Un inventario risulta utile anche per la possibilità di poter specificare dei parametri di configurazione per il singolo host, permettendo di poter salvare l'indirizzo IP associato all'host, o la porta SSH nel caso in cui essa non sia sulla porta predefinita. Un'esempio più completo, nel formato YAML, è dato da:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
        jumper:
          ansible_port: 5555
          ansible_host: 192.0.2.50
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
    east:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    west:
      hosts:
        bar.example.com:
        three.example.com:
    prod:
      children:
        east:
    test:
      children:
        west:
```

La possibilità di specificare parametri non standard risulta particolarmente utile nel contesto della tesi, in quanto Ansible prova ad utilizzare di default OpenSSH come metodo di connessione. Quando si prova ad effettuare l'accesso con password viene utilizzato *sshpass*, il quale non è completamente supportato da BSD. Poichè la vNSF che si intende realizzare ha come sistema operativo di base un sistema BSD è risultato opportuno cambiare il metodo di trasporto a *paramiko*. Inoltre è stato necessario impostare l'interprete Python, in quanto la locazione di default sui sistemi Unix/Linux è /usr/bin/python, mentre sui sistemi BSD potrebbe essere differente (e.g. /usr/local/bin/python2.7).

3.4.2 Playbook

Un playbook rappresenta una lista di task da eseguire su determinati gruppi di host, in modo da evitare l'utilizzo di comandi Ansible ad-hoc per ogni singolo task. Esso sfrutta le informazioni contenute nell'inventario ed i moduli messi a disposizione da Ansible per stabilire su quali nodi effettuare tali operazioni. I playbook forniscono uno strumento molto potente che rappresenta

la base per effettuare configurazioni con Ansible. Oltre alle configurazioni più semplici, questo strumento permette di gestire qualunque processo complesso, sincrono o asincrono che sia, o che preveda uno scambio di messaggi tra i nodi dell'infrastruttura. I playbook vengono specificati in YAML, e sono stati progettati per essere leggibili e semplici da modificare, in quanto il linguaggio in cui vengono specificati contiene una sintassi ristretta e semplice da utilizzare. Ogni playbook è costituito da una o più azioni (comunemente riferite con il termine 'plays'). Una azione definisce una sequenza di task, che comunemente coincide con una specifica chiamata ad un modulo Ansible, su un gruppo di host (definito nell'inventario). Oltre alla scelta dell'host su cui eseguire una azione, è possibile specificare anche l'utente remoto, definendolo a livello di host o a livello di singolo task. Un esempio di playbook è il seguente:

```
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
    tags:
      - checkStatus
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

In questo caso è possibile individuare due azioni, la prima eseguita per il gruppo di host *webservers*, la seconda per il gruppo *databases*, definiti nell'esempio nel paragrafo 3.4.1. Per entrambe le azioni, i task verranno eseguiti come utente root, richiamando i moduli Ansible definiti nel playbook (e.g. yum, service, template). I task vengono eseguiti in modo sequenziali ed in modo identico su tutti gli host definiti all'interno del gruppo, differenziandosi solo dalle variabili definite per ogni host. Si ha la possibilità di definire dei tag, per eseguire o escludere delle operazioni dall'esecuzione di una azione. Le operazione devono essere idempotenti e funzionanti su tutti i nodi nell'host, in quanto se una operazione fallisce su un nodo, esso verrà escluso da tutto il resto della catena. Ogni task viene definito da un nome, stampato ad output con le informazioni relative al task, ed un modulo, che accetta parametri nel formato key=value. Ansible mette a disposizioni un gran numero di moduli, con cui è possibile effettuare ogni tipo di configurazione sui nodi gestiti, oltre a dare la possibilità di definire moduli ad-hoc per degli scopi specifici. Un playbook può essere eseguito utilizzando il comando *ansible-playbook* e specificando il playbook da utilizzare. Ad esempio, per eseguire il playbook playbook.yaml:

```
ansible-playbook playbook.yaml
```

Inoltre è possibile specificare eventuali password per esecuzione di comandi da amministratore, utilizzando i parametri `--ask-become-pass` o `-K`.

3.5 Architettura della soluzione

Per la realizzazione della vNSF sono stati analizzati nella sezione 2.4.2, in particolare in Fig. 2.18, i moduli principali per la configurazione del packet filter nel contesto desiderato. Tale vNSF è progettata per far parte di un NS composto da una o più VNF e che sia in grado di fornire un servizio di sicurezza. In questo contesto viene creato un NS che prevede l'utilizzo di due VNF a singola interfaccia da collegare al packet filter realizzato, come mostrato in Fig. 3.13. Questa architettura verrà utilizzata per verificare il corretto filtraggio del traffico alle estremità della vNSF. Per maggiori informazioni sui risultati del test si rimanda al capitolo 4. Nel NS

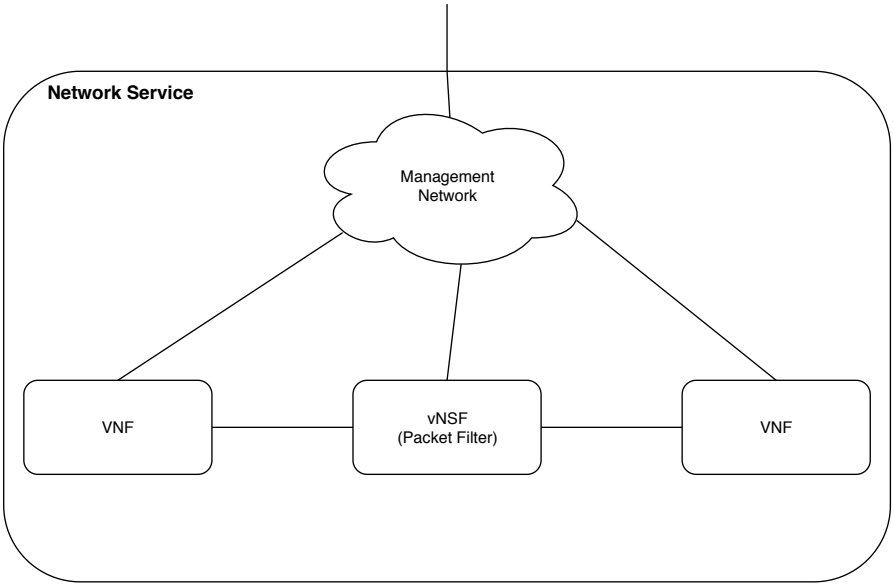

Figura 3.13. Architettura del NS

realizzato il packet filter è collegato a due VNF che rappresentano un'istanza di FreeBSD senza alcuna funzionalità specifica. È possibile assegnare un numero arbitrario di interfacce alla vNSF specificandolo nel descrittore della funzione di rete ed effettuare il collegamento con un numero arbitrario di VNF per implementare una qualunque catena di servizi. Andiamo ora ad analizzare come gli strumenti descritti interagiscano con la vNSF realizzata nelle fasi di istanziazione e configurazione del packet filter.

3.5.1 Interazione dei componenti

Vengono adesso analizzati i due flussi principali riguardanti il ciclo di vita della VNF, ossia la fase di istanziazione della VNF e la fase di configurazione del packet filter, inteso sia come configurazione iniziale sia come gestione delle regole impostate sul packet filter durante il suo ciclo di vita. Queste due fasi sono successive alla fase di onboarding della VNF e del relativo NS. Nell'architettura di OSM tale operazione viene inoltrata direttamente dal NBI verso i servizi di storage e database, sulle quali vengono effettuate le operazioni *Create*, *Read*, *Update*, *Delete* (CRUD) in modo che le informazioni possano essere gestite all'interno di OSM. Tale operazione quindi, come mostrato in Fig. 3.14 non viene processata dal resto dei componenti dell'architettura OSM non essendo inoltrata sul bus Kafka. Questa fase viene svolta interamente all'interno di OSM e non coinvolge attori esterni.

Istanziazione della VNF

Una volta che la fase di onboarding è terminata, è possibile istanziare il NS e tutte le VNF che esso include. Questa fase coinvolge anche il NFVI poiché vengono allocate le risorse. La sequenza di operazioni che portano all'istanziazione del servizio di rete viene mostrata in Fig. 3.15.

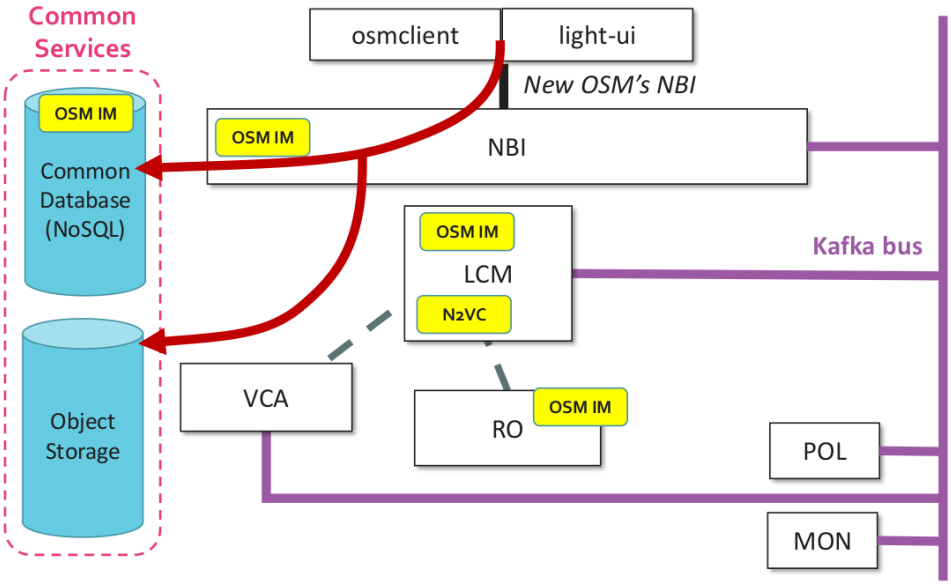

Figura 3.14. Fase di Onboarding in OSM

Il processo inizia con la richiesta di istanziazione di un NS presente nel database di OSM. Tale richiesta può essere effettuata tramite il client messo a disposizione da OSM, oppure sfruttando l'interfaccia grafica. La richiesta viene quindi inoltrata alla NBI, la quale dopo aver creato un record relativo al NS che si intende istanziare all'interno del database comune di OSM, invia un messaggio utilizzando il bus Kafka, del tipo *Instantiate NS_id*.

Il messaggio inviato dalla NBI viene consumato dal modulo LCM, il quale una volta ricevuta la richiesta invia al modulo RO le informazioni relative al NS che si intende istanziare. Il modulo RO, grazie alle informazioni contenute nei descrittori forniti in fase di onboarding definisce le risorse necessarie da allocare sul NFVI, definendo il numero di istanze virtuali necessarie per le VNF ed il numero di VL da utilizzare.

A questo punto il modulo RO si occupa di interagire con il VIM fornito da OpenStack per istanziare tali risorse. Si vanno quindi ad utilizzare le API messe a disposizione dai servizi di OpenStack, come Neutron per la configurazione delle reti virtuali o Nova per la creazione delle macchine virtuali su cui verranno eseguite le funzioni di rete. Durante questa fase vengono sfruttate le funzionalità messe a disposizione da bsd-cloudinit per poter creare un'istanza di FreeBSD con le dimensioni specificate nel descrittore della vNSF di PF.

Configurazione della VNF

Terminata la fase di istanziazione la VNF è pronta per essere configurata. Tale fase include uno step di configurazione iniziale ed una fase di gestione delle regole impostate su PF durante il ciclo di vita della vNSF.

Il flusso seguito per la configurazione iniziale viene mostrato in Fig. 3.16. Durante questa fase entra in gioco il modulo VCA, in quanto in esso è contenuto il Juju controller atto alla gestione della configurazione della vNSF. Tali operazioni avvengono cronologicamente successivamente al flusso mostrato per l'istanziazione delle risorse. Il modulo LCM invia la richiesta di effettuare il deploy delle VNF al VCA. Nel caso specifico le VNF poste ai margini della vNSF di PF non richiedono nessuna configurazione da parte di Juju, motivo per cui per esse non viene effettuata alcuna operazione dal Juju controller. Per gestire la vNSF viene creato un *proxy charm*. Esso risulta nella creazione di un container LXD nel modulo VCA che fornisce un servizio legato alla vNSF istanziata e che realizza le operazioni di configurazione fornite nel Juju charm.

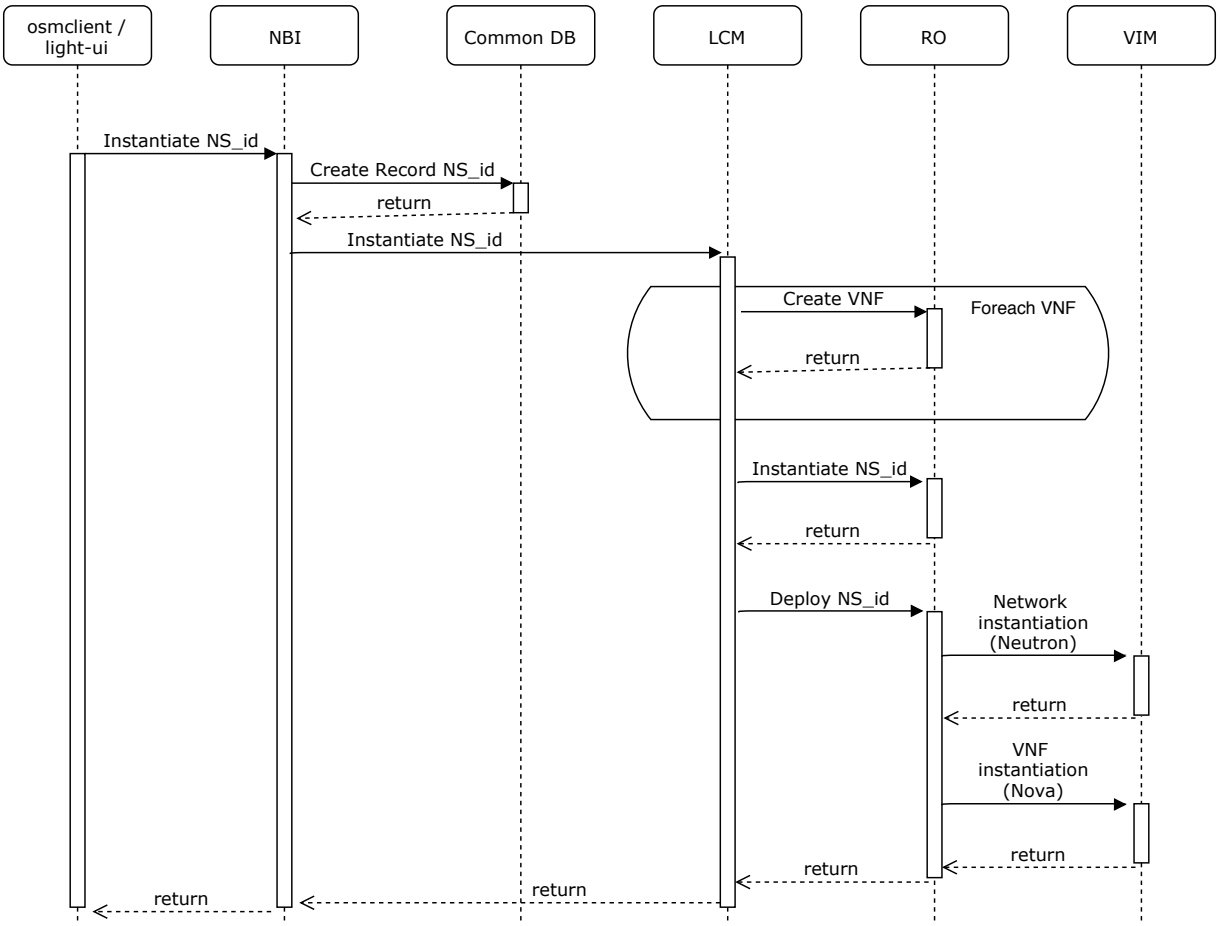

Figura 3.15. Fase di Instanziazione di un NS

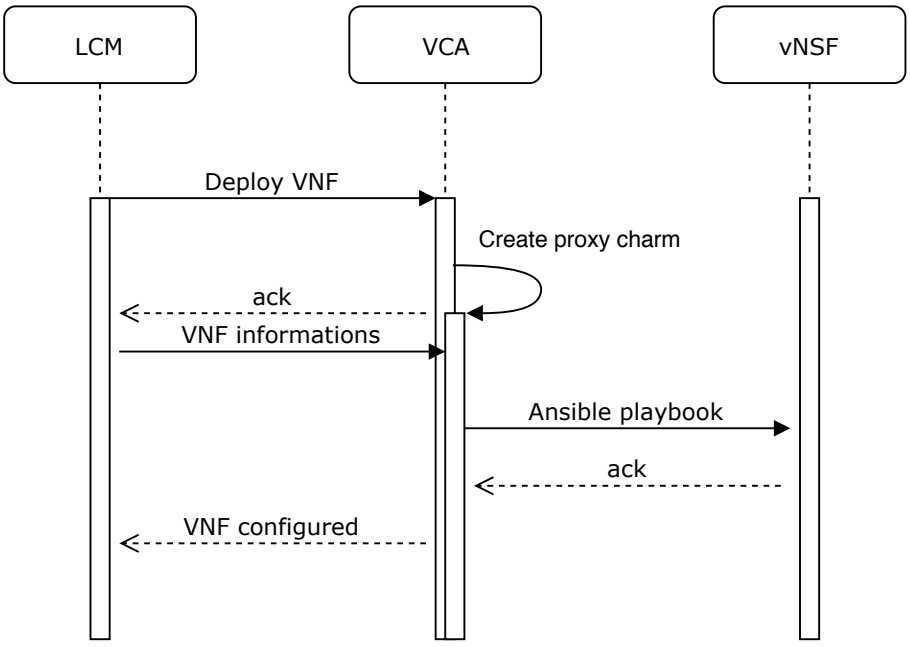

Figura 3.16. Configurazione iniziale della vNSF

Nel momento in cui il proxy charm viene creato esso riceve le informazione relative alla funzione di rete da gestire (i.e. indirizzo dell'interfaccia di management, credenziali SSH). A questo punto esso procede con le operazioni di configurazione iniziali definite per la vNSF. Nel caso specifico il charm mette a disposizione un layer Ansible che permette l'esecuzione di un playbook ad-hoc definito per l'inizializzazione dell'immagine di FreeBSD utilizzata per lo scopo di tesi. La scelta di utilizzare lo strumento Ansible deriva dalla semplicità di utilizzo e dall'efficienza messa a disposizione da questo strumento di configurazione, il quale non richiede l'istallazione di nessun agent specifico sulla VNF istanziata. Al termine dell'esecuzione del playbook sulla vNSF sono state installate le dipendenze necessarie per l'esecuzione di PF ed il modulo di packet filtering viene inizializzato e caricato. Inizialmente non sono presenti regole su PF, per cui qualunque tipo di traffico è consentito. Terminata la configurazione iniziale la vNSF risulta in uno stato *Ready* ed è pronta a gestire operazioni di configurazione delle politiche di filtraggio. Juju resta in attesa di richieste di esecuzione delle action definite dal charm.

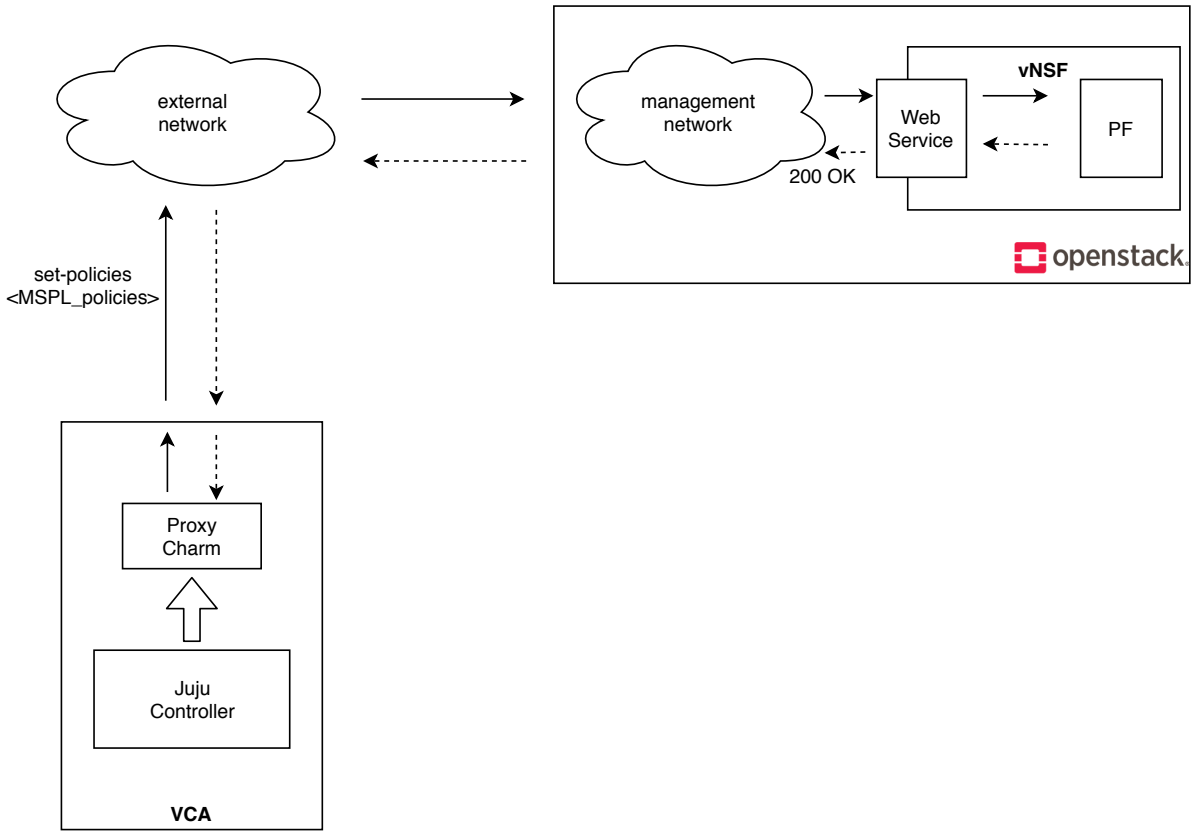

Figura 3.17. Configurazione della vNSF attraverso il web service

Le richieste vengono avviate utilizzando il client messo a disposizione da Juju. A questo punto il Juju controller invia la richiesta al proxy charm incaricato di gestire la vNSF, il quale elabora la richiesta e configura PF. Nell'architettura utilizzata nel contesto della tesi, non è stato utilizzato un sistema di *refinement* delle policy da HSPL a MSPL. Per questo motivo è necessario specificare le policy in formato MSPL quando si intende effettuare operazioni di aggiunta delle policy all'interno del packet filter. In un contesto di utilizzo reale, viene fornito un componente in grado di fornire policy di sicurezza specificate in linguaggio MSPL, come ad esempio il *Reccomendation and Remediation Engine* nel progetto SHIELD, e di inviare richieste di configurazione con policy specificate in tale formato.

Per la gestione di tali richieste, è stato implementato un web service in ascolto sull'interfaccia esposta sulla rete di management. Tale servizio è responsabile di ricevere le richieste inviate da Juju ed elaborarle. Esso effettua la fase di traduzione della policy dal formato MSPL alla configurazione specifica per PF nel caso in cui si riceva la richiesta di aggiungere una nuova policy di sicurezza. Maggiori informazioni sull'API esposta dal web service vengono fornite nel capitolo

di analisi dell'implementazione della soluzione (Cap. 4). In Fig. 3.17, a titolo di esempio, viene mostrato lo scambio di messaggi che avviene nel caso d'uso in cui si vogliano aggiungere nuove politiche al modulo di packet filtering.

Capitolo 4

Implementazione e Test

In questo capitolo vengono discusse le scelte implementative legate alla realizzazione della vNSF. Viene affrontato in modo dettagliato il modulo di inizializzazione della VNF per poi descrivere l'API messa a disposizione per la configurazione del packet filter dal lato del Juju charm e come tali richieste vengano gestite dalla funzione virtuale per la configurazione di PF. Il capitolo presenta inoltre alcuni test effettuati sulla vNSF. È presente un test funzionale atto a validare il funzionamento della vNSF in termini di configurazione ed un test di performance per valutarne le prestazioni.

4.1 Implementazione

L'implementazione della vNSF è stata suddivisa logicamente in due blocchi principali. È infatti possibile individuare un blocco gestito ed utilizzato dal gestore della vNSF, ed un blocco atto a gestire le richieste dal lato dell'istanza virtuale.

4.1.1 Modulo di configurazione sul VNF Manager

Il modulo descritto in questa sezione viene utilizzato da OSM, fornendo metodi implementati per poter eseguire operazioni di configurazione ed inizializzazione della vNSF. Come già discusso precedentemente, OSM prevede all'interno del suo stack di funzionamento l'utilizzo di Juju. La fase in analisi dunque si incentra sulla definizione del Juju charm utilizzato per gestire la funzione di sicurezza.

Il Juju charm è stato organizzato principalmente in due blocchi differenti, mostrati in Fig. 4.1. Il primo blocco prevede l'utilizzo di Ansible. Tale strumento risulta essere tra i più utilizzati per la configurazione delle istanze in ambito cloud. Inoltre fornisce una soluzione efficiente e facilmente estendibile ed adattabile grazie alla semplicità pensata per tale strumento. Come già descritto inoltre, non richiedendo agent aggiuntivo sull'istanza da configurare, non appesantisce la soluzione realizzata per la vNSF. L'utilizzo di tale strumento è stato reso possibile anche dal lavoro effettuato dai developer di OSM che hanno fornito una soluzione che permetta di integrare facilmente Ansible all'interno di un charm. Viene infatti fornito un layer, nello specifico denominato `ansible-base`, che viene incluso all'interno del charm affinchè Ansible possa essere utilizzato dal proxy charm istanziato. Le informazioni utilizzate da Ansible per poter configurare l'istanza, quali indirizzo dell'interfaccia di management della vNSF, nome utente e password, vengono gestiti direttamente all'interno del charm. Juju riesce infatti ad accedere alle informazioni di configurazione utilizzate ottenendole dal resto dell'infrastruttura di OSM (e.g. accedendo alle informazioni nel descrittore della VNF). Tale meccanismo permette dunque di definire ed eseguire i playbook Ansible all'interno del charm. Questo viene sfruttato nel contesto del lavoro realizzato tramite l'implementazione di un playbook. In particolare viene gestita la fase di inizializzazione, nella quale vengono installate le dipendenze necessarie e viene avviato il servizio atto a gestire le richieste di configurazione sull'istanza virtuale. Il modulo realizzato con Ansible

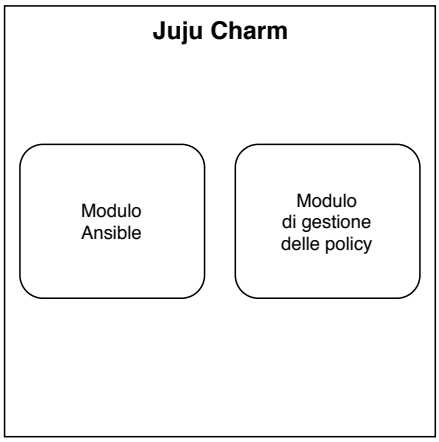

Figura 4.1. Blocchi del Juju charm

è inoltre responsabile dell'esecuzione di operazioni che gestiscono il ciclo di vita del web service di configurazione anche successivamente alla fase di inizializzazione. Vengono a tal proposito implementate le funzionalità:

- **start:** avvia un'istanza del web service sulla porta specificata tramite il parametro `api_port`, applicando al packet filter lo stato impostato sul file di configurazione utilizzato da PF, per il quale si forniranno maggiori dettagli nella sezione successiva;
- **stop:** interrompe il web service in ascolto sulla porta specificata tramite il parametro `api_port`, ripristina lo stato del packet filter allo stato iniziale, annullando di fatto la configurazione corrente;
- **restart:** invoca in sequenza le operazioni di stop e start, utilizzando anche in questo caso il parametro `api_port`;

Il secondo blocco individuato in Fig. 4.1, è invece responsabile di rispondere alle richieste relative alla gestione delle policy applicate al packet filter. Tali richieste sono effettivamente gestite dal web service in esecuzione sulla vNSF. Il charm si limita ad inviare le richieste all'API esposta dal servizio web citato tramite l'utilizzo di *curl* e ne visualizza i risultati. Per il momento ci si limita dunque a dare una descrizione dell'API che il charm realizza, per capirne meglio il funzionamento nella sezione successiva. Le *action* fornite dal charm sono:

- **get-policies:** effettua la richiesta per ottenere le regole correntemente impostate sul packet filter.
- **set-policies:** effettua la richiesta di aggiunta di nuove regole all'interno del packet filter. Utilizza il parametro `policies`, in cui vengono specificate le policy di sicurezza che si desidera applicare in formato MSPL.
- **delete-policies:** effettua la richiesta per eliminare le regole impostate correntemente sul packet filter.
- **delete-policy:** effettua la richiesta per cancellare una singola policy impostata sul packet filter. Utilizza il parametro `policy`, definito come intero, che identifica il numero della policy che si desidera eliminare.
- **set-quick:** effettua la richiesta per impostare un comportamento di default per la creazione di nuove regole. Come già discusso è possibile specificare all'interno di una regola PF la keyword *quick* per fare in modo che il packet filter non effettui controlli con regole successive se la regola trovata contiene tale parola chiave. Utilizza il parametro `quick`, definito come valore booleano, che indica se utilizzare *quick* all'interno delle regole in cui tale valore non è direttamente specificato.

- **get-quick:** effettua la richiesta per ottenere informazioni relative al comportamento di default utilizzato per la parola chiave *quick*.

4.1.2 Web Service di configurazione della vNSF

Si procede con la descrizione del web service in esecuzione sulla vNSF che si occupa della gestione delle politiche del packet filter. Il servizio web è stato implementato utilizzando il linguaggio Python. Viene in particolare utilizzato il framework Flask [37], il quale fornisce una soluzione messa a disposizione per costruire servizi web leggeri in modo rapido ed efficiente. Il web service realizzato offre un'interfaccia organizzata in modo semplice, motivo per cui si è scelto di utilizzare tale framework.

Il web service è stato organizzato in più moduli logici, mostrati in Fig. 4.2 ed individuati da:

- **Web Service interface:** rappresenta il modulo che espone l'API sull'interfaccia di gestione.
- **XML Validator:** fornisce il modulo atto a validare il formato della richiesta di aggiunta di nuove policy.
- **PF function wrapper:** fornisce un contenitore che effettua un incapsulamento delle chiamate di gestione di PF in funzioni invocabili nel web service Python.
- **MSPL translator:** fornisce il modulo di traduzione della policy MSPL, mappando le richieste ricevute in oggetti che incapsulano la configurazione di basso livello di PF.
- **PF Wrapper:** rappresenta il cuore del web service, gestendo di fatto le richieste ricevute tramite l'interfaccia del web service e comunicando con i moduli **MSPL Translator** e **PF function wrapper** per applicare la configurazione richiesta.

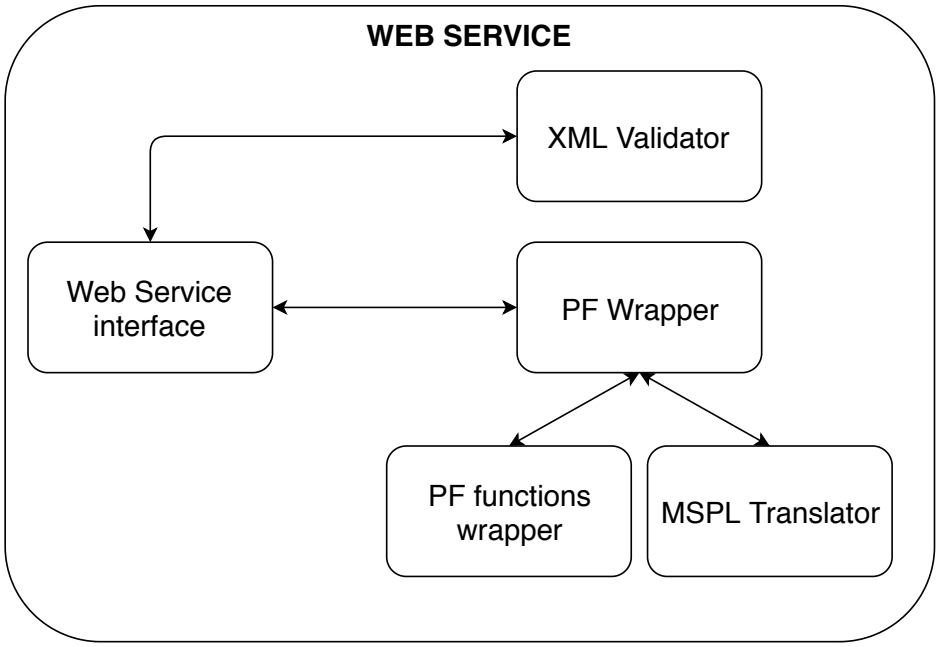

Figura 4.2. Struttura del servizio web di gestione della vNSF

Viene ora fornita una panoramica dell'interfaccia messa a disposizione, per poi affrontare l'iterazione avuta da questi componenti per i diversi casi. L'API fornita è data da:

- `http://Web_Service_address:API_PORT/setRules/v1`: contattato tramite metodo POST, imposta le regole richieste sul packet filter. Viene utilizzato dall'azione `set-policies` dal Juju charm.

- `http://Web_Service_address:API_PORT/getRules/v1`: contattato tramite metodo GET, restituisce le regole impostate sul packet filter. Viene utilizzato dall'azione `get-policies` del Juju charm.
- `http://Web_Service_address:API_PORT/flushRules/v1`: contattato con il metodo DELETE, elimina tutte le regole impostate sul packet filter. Viene utilizzato dall'azione `delete-policies` del Juju charm.
- `http://Web_Service_address:API_PORT/flushRules/v1/<rule_id>`: contattato tramite metodo DELETE, elimina la regola identificata dal numero corrispondente al valore di `rule_id`. Viene utilizzato dall'azione `delete-policy` del Juju charm.
- `http://Web_Service_address:API_PORT/setQuick/v1`: contattato tramite metodo POST, imposta il comportamento di default da utilizzare per l'aggiunta di nuove regole. Viene utilizzato dall'azione `set-quick` del Juju charm.
- `http://Web_Service_address:API_PORT/getQuick/v1`: contattato tramite metodo GET, fornisce informazioni sul comportamento di default da utilizzare nel caso in cui il parametro quick non sia specificato nella regola. Viene utilizzato dall'azione `get-quick` del Juju charm.

4.1.3 Casi d'uso del Web Service

Si procede con una descrizione del flusso previsto per i casi d'uso principali messi a disposizione dall'API esposta dal servizio web. Il caso d'uso previsto per la gestione della parola chiave `quick` viene escluso da tale analisi. Esso prevede infatti un flusso di esecuzione molto semplice, in cui è previsto l'aggiornamento di una variabile contenuta all'interno del servizio web per tracciare lo stato corrente.

Aggiunta di policy

Il flusso di esecuzione per il caso d'uso in analisi viene riassunto in Fig. 4.3. Il flusso inizia

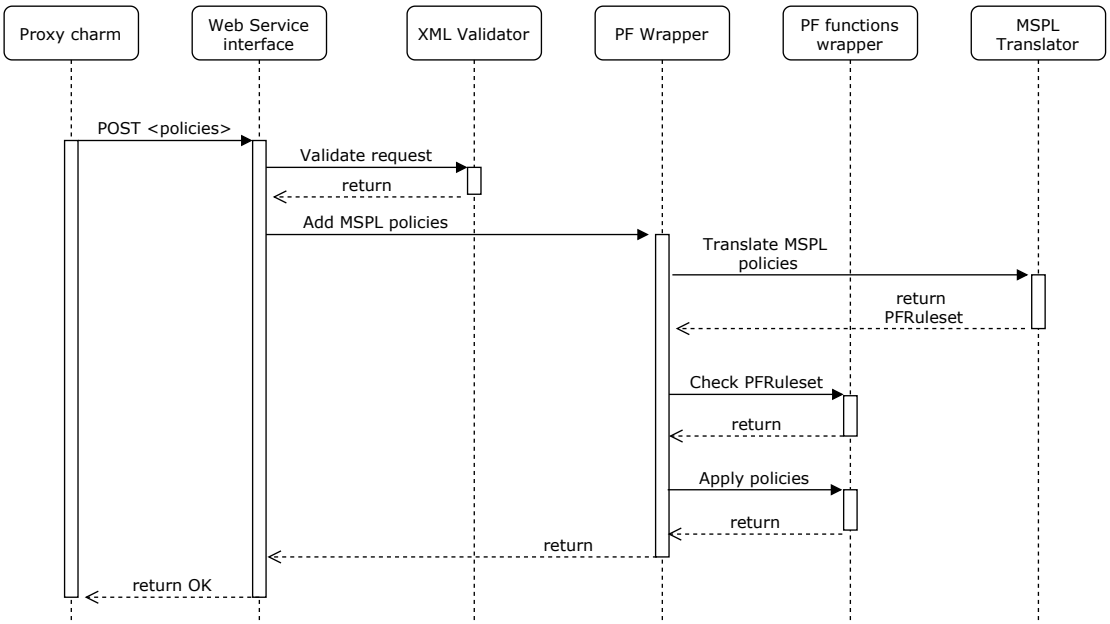

Figura 4.3. Flusso di esecuzione per l'aggiunta delle regole

tramite una chiamata POST all'indirizzo `http://Web_Service_address:API_PORT/setRules/v1` contenente le politiche in formato MSPL da aggiungere. Nell'esempio rappresentato nel diagramma

si considera il caso in cui la richiesta sia inviata direttamente dal proxy charm. Per il flusso di esecuzione si individuano i seguenti passaggi:

1. La richiesta viene intercettata dall'interfaccia che procede a validare le politiche contenute nel corpo della richiesta. L'MSPL ricevuto viene validato dall'apposito modulo, il quale confronta la richiesta con un XML Schema contenente la definizione delle politiche MSPL che è possibile ricevere. Si è deciso di gestire la possibilità di specificare più politiche di configurazione all'interno di una singola richiesta.
2. Dopo aver validato le politiche ricevute, la richiesta viene inoltrata al PF Wrapper, il quale si occupa di gestire l'aggiunta di tali politiche.
3. Il modulo PF Wrapper invia le politiche in formato MSPL all'apposito blocco di traduzione. L'esecuzione di tale operazione restituisce un oggetto di tipo PFRuleset. L'oggetto PFRuleset è composto da più oggetti di tipo PFRule, nel quale si vanno a mappare le richieste MSPL. Tali oggetti forniscono la rappresentazione della regola PF di basso livello e vengono utilizzati per salvare la regola sul file di configurazione utilizzato per le regole del packet filter.
4. Il modulo PF Wrapper, tramite l'utilizzo delle funzioni di gestione di PF effettua un controllo di tipo sintattico sul PFRuleset che si intende aggiungere.
5. Dopo che le regole sono state verificate, esse vengono copiate sul file di configurazione utilizzato per gestire le regole di PF. A questo punto le nuove politiche vengono aggiunte al packet filter tramite l'utilizzo dei comandi di gestione.
6. Una volta che le politiche sono state aggiunte viene inviata una risposta dall'interfaccia del servizio web per notificare l'operazione avvenuta.

Il caso mostrato prevede che le operazioni di validazione avvengano con successo. In caso contrario il web service invia in risposta un messaggio di errore per notificare l'avvenuto fallimento con alcuni dettagli relativi all'errore avvenuto.

Ottenimento delle policy impostate

L'interazione dei componenti per ottenere le politiche impostate sul packet filter viene mostrato in Fig. 4.4. Questo flusso inizia con una chiamata di tipo GET, senza parametri aggiuntivi, all'indirizzo `http://Web_Service_address:API_PORT/getRules/v1`. In questo caso il flusso risulta più semplice:

1. La richiesta viene ricevuta dall'interfaccia del servizio web ed inoltrata al modulo PF Wrapper.
2. La richiesta viene gestita dal PF Wrapper, il quale si occupa di recuperare le regole correntemente impostate sul packet filter. Tale blocco si occupa di mappare i campi delle regole di PF in formato JSON in modo da renderle più leggibili ed analizzabili. Le singole regole vengono infine associate ad un valore intero che indica il numero della regola sul packet filter. Tale valore potrà essere utilizzato nel caso d'uso dell'eliminazione della regola.
3. L'interfaccia del web service, ricevuta la rappresentazione JSON delle regole impostate, invia il messaggio di risposta. Nel caso in cui non ci siano regole impostate al momento della richiesta si invia in risposta il messaggio "No rules".

Rimozione delle policy

Il flusso di rimozione delle policy viene mostrato in Fig. 4.5. Il flusso viene attivato tramite una chiamata all'indirizzo `http://Web_Service_address:API_PORT/flushRules/v1/<rule_id>`, utilizzando il metodo DELETE. Per questo caso d'uso vengono individuati i seguenti passaggi:

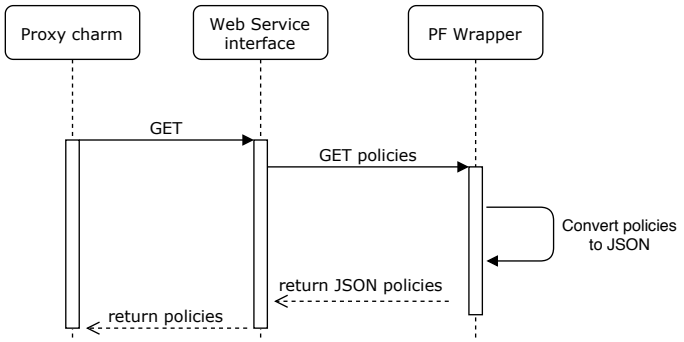

Figura 4.4. Flusso di esecuzione per l'ottenimento delle regole

1. La richiesta viene ricevuta dall'interfaccia del servizio web ed inoltrata al modulo PF Wrapper. Nella richiesta viene specificato un intero, identificato da `rule_id`, atto ad indicare il numero associato alla regola che si intende eliminare.

2. Il blocco PF Wrapper gestisce l'eliminazione della regola specificata rimuovendo la riga corrispondente dal file di configurazione utilizzato dal packet filter.

3. Il PF Wrapper utilizza le funzionalità incapsulate nel blocco PF Function Wrapper per aggiornare la configurazione del packet filter.

4. Terminate le operazione l'interfaccia risponde con un messaggio atto a notificare che l'operazione desiderata è stata eseguita con successo.

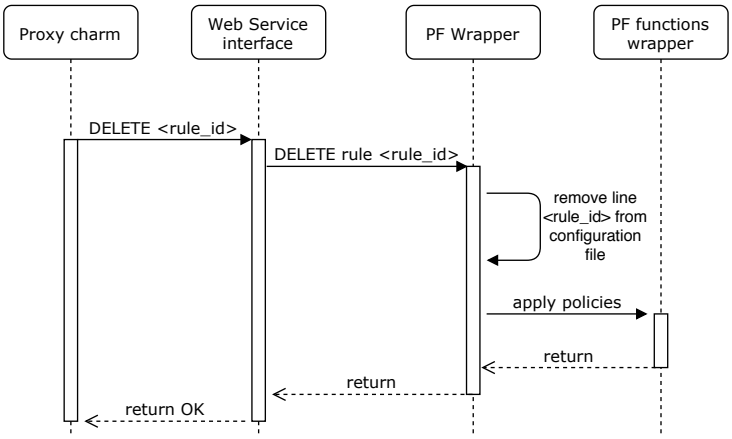

Figura 4.5. Flusso di esecuzione per l'eliminazione delle regole

Il flusso mostrato rappresenta l'eliminazione della singola regola. Come descritto in precedenza risulta possibile richiedere l'eliminazione di tutte le regole impostate sul packet filter. In quest'ultimo caso si utilizza l'indirizzo `http://Web_Service_address:API_PORT/flushRules/v1`, contattato tramite metodo DELETE. Questo caso risulta differente dal flusso illustrato nel punto in cui viene aggiornato il file di configurazione. In questo caso infatti, viene semplicemente eliminato l'intero contenuto del file di configurazione, per poi resettare le regole impostate su PF.

4.2 Test

4.2.1 Test funzionale

Il test descritto in questa sezione viene realizzato al fine di mostrare il corretto funzionamento della vNSF. L'obiettivo è verificare che in risposta ad alcuni comandi di configurazione inviati il packet filter risponda nel modo corretto, bloccando o consentendo il traffico a seconda dei casi.

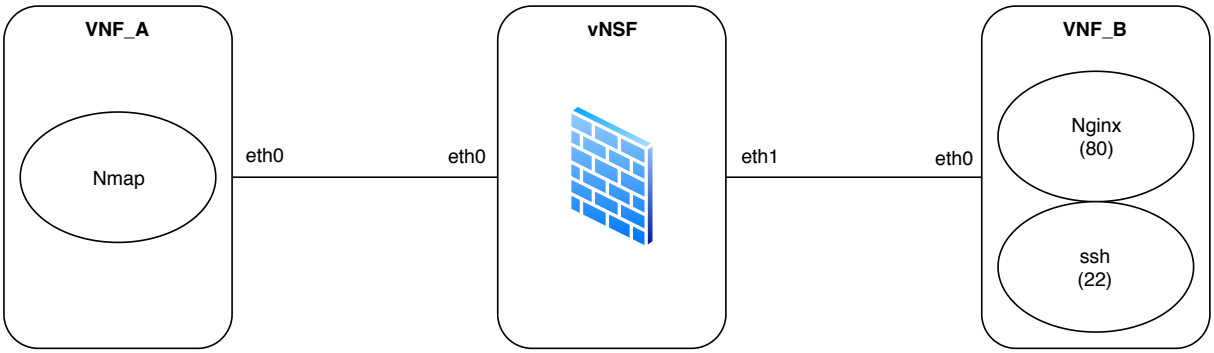

Figura 4.6. Architettura utilizzata nel test funzionale

L'architettura utilizzata per il test viene mostrata in Fig. 4.6. Come mostrato in figura, nel test si fa utilizzo dei seguenti strumenti:

- **Nmap [36]:** fornisce uno strumento di rete open source ampiamente utilizzato per effettuare network discovery. Fornisce la soluzione ideale per eseguire la scansione delle porte all'interno del test per determinare quali di esse sono esposte.
- **Nginx:** web server esposto sulla porta 80.
- **SSH:** servizio di connessione esposto sulla porta 22.

l'obiettivo è utilizzare Nmap per verificare, a partire dall'istanza riferita come VNF_A, quali porte dell'istanza VNF_B risultino raggiungibili. A tale scopo sono stati esposti i servizi Nginx ed SSH sulla VNF_B. La configurazione per le istanze utilizzate in fase di test viene mostrata in Tabella 4.1.

Istanza	vCPU	RAM	OS
VNF_A	1	512 MB	FreeBSD 11.1
vNSF	1	512 MB	FreeBSD 11.1
VNF_B	1	512 MB	FreeBSD 11.1

Tabella 4.1. Configurazione delle istanze nel test funzionale

Durante la fase di test vengono sperimentate le politiche ACCEPT e REJECT. Al variare delle configurazioni, il risultato atteso dall'esecuzione della scansione delle porte rientra nei seguenti casi:

- open: la porta risulta aperta e raggiungibile. Risultato atteso in caso di ACCEPT del traffico.

- closed: la porta risulta chiusa e dunque non raggiungibile. Risultato atteso in caso di REJECT sulla porta.

Nella definizione delle regole viene utilizzato l'indirizzo IP assegnato alle istanze, risulta dunque utile effettuare una mappatura tra gli indirizzi assegnati e l'immagine relativa all'architettura (Fig. 4.6). Essi sono:

- VNF_A: indirizzo 22.22.22.35 assegnato all'interfaccia eth0.
- vNSF: indirizzo 22.22.22.14 assegnato all'interfaccia eth0, indirizzo 11.11.11.32 assegnato all'interfaccia eth1.
- VNF_B: indirizzo 11.11.11.35 assegnato all'interfaccia eth1.

Per l'utilizzo degli strumenti in esecuzione su VNF_A e VNF_B viene utilizzata la console messa a disposizione da OpenStack (si faccia riferimento all'Appendice A). Invece, si procede con l'iniezione delle regole sulla vNSF tramite l'utilizzo del Juju charm realizzato, per sperimentare che la configurazione del packet filter avvenga in modo opportuno a partire dalla regola MSPL.

Regole non impostate

La situazione iniziale prevede che entrambi i servizi esposti sulla VNF_B siano raggiungibili dalla VNF_A. Si ha infatti una situazione iniziale in cui l'istanza del packet filter è appena stata creata e sulla quale non sono inizialmente impostate regole di filtraggio. Per verificare tale situazione si procede con l'eseguire tramite Nmap il comando

```
$ nmap 11.11.11.35
```

l'output in questo caso risulta il seguente

```
Nmap scan report for 11.11.11.35
Host is up (0.00050s latency).
Not shown: 998 closed ports
PORT STATE SERVICE
22/tcp open ssh
80/tcp open http
```

per verificare inoltre il corretto funzionamento dell'API esposta del web service viene effettuata una chiamata per ricevere le policy impostate sul packet filter. La risposta JSON in questo caso risulta

```
{"message":"No rules","status":200}
```

Impostazione delle regole di blocco

Si procede con l'impostare le regole di blocco sul packet filter per impedire che le porte esposte dai servizi siano raggiunte dalla VNF_A e che dunque risultino chiuse utilizzando Nmap. Le regole nel formato MSPL vengono iniettate tramite l'utilizzo del proxy charm. Non avendo a disposizione uno strumento che generi automaticamente la policy MSPL a partire da una specifica nel formato HSPL, si procede con la creazione manuale della politica. Per il blocco di regole viene utilizzato il seguente input

```
<mspl-set xmlns="http://security.polito.it/shield/mspl"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://security.polito.it/shield/mspl
    mspl.xsd"><it-resource id="vNSF2"><configuration
    xsi:type="filtering-configuration">
<rule>
```

```
<priority>1</priority><action>reject</action>
<condition><packet-filter-condition><direction>inbound</direction>
<source-address>22.22.22.35</source-address><destination-port>22</destination-port>
<protocol>TCP</protocol></packet-filter-condition></condition>
</rule>
<rule>
<priority>2</priority><action>reject</action>
<condition><packet-filter-condition><direction>inbound</direction>
<source-address>22.22.22.35</source-address><destination-port>80</destination-port>
<protocol>TCP</protocol></packet-filter-condition></condition>
</rule>
<rule>
<priority>2</priority><action>reject</action>
<condition><packet-filter-condition><direction>inbound</direction>
<source-address>22.22.22.35</source-address><destination-port>5000</destination-port>
<protocol>TCP</protocol></packet-filter-condition></condition>
</rule>
</configuration></it-resource></mspl-set>
```

iniettando le politiche illustrate si procede al blocco delle porte 22, 80 e 5000. Il blocco di tre porte risulterà utile per analizzare il funzionamento dell'eliminazione delle politiche di sicurezza. L'output fornito da Nmap, con tali regole impostate, risulta come previsto

```
Starting Nmap 7.01 ( https://nmap.org ) at 2019-11-08 21:39 UTC
Nmap scan report for 11.11.11.35
Host is up (0.00034s latency).
All 1000 scanned ports on 11.11.11.35 are closed
```

inoltre la chiamata all'API per visualizzare le regole impostate restituisce il seguente output

```
{
"0":{"action":"reject",
     "destination-port":"22",
     "direction":"inbound",
     "dst":"any",
     "protocol":"tcp",
     "quick":"True",
     "src":"22.22.22.35"},
"1":{"action":"reject",
     "destination-port":"80",
     "direction":"inbound",
     "dst":"any",
     "protocol":"tcp",
     "quick":"True",
     "src":"22.22.22.35"},
"2":{"action":"reject",
     "destination-port":"5000",
     "direction":"inbound",
     "dst":"any",
     "protocol":"tcp",
     "quick":"True",
     "src":"22.22.22.35"}
}
```

le regole risultano dunque, anche nel caso corrente, correttamente impostate e visualizzate.

Eliminazione delle regole

Si procede con la verifica della funzionalità di eliminazione della singola regola. Ciò avviene invocando l'azione `delete-policy` messa a disposizione dal charm. In particolare viene eliminata la prima regola impostata, relativa al blocco sul servizio SSH. L'esecuzione di Nmap porta quindi al seguente output

```
Starting Nmap 7.01 ( https://nmap.org ) at 2019-11-08 21:47 UTC
Nmap scan report for 11.11.11.35
Host is up (0.00068s latency).
Not shown: 999 closed ports
PORT STATE SERVICE
22/tcp open ssh
```

le policy impostate risultano invece

```
{
"0":{"action":"reject",
     "destination-port":"80",
     "direction":"inbound",
     "dst":"any",
     "protocol":"tcp",
     "quick":"True",
     "src":"22.22.22.35"},
"1":{"action":"reject",
     "destination-port":"5000",
     "direction":"inbound",
     "dst":"any",
     "protocol":"tcp",
     "quick":"True",
     "src":"22.22.22.35"}
}
```

Verificata l'eliminazione della singola policy, si va infine a verificare l'azione `delete-policies`. Essa a differenza della precedente elimina tutte le politiche impostate su PF. Tale operazione riporta quindi il packet filter allo stato iniziale, riproducendo i risultati ottenuti durante il test effettuato senza politiche impostate.

4.2.2 Test di performance

Lo scopo di questo test è verificare le prestazioni che la vNSF riesce a raggiungere misurando le variazioni di throughput dovute all'inserimento del nodo contenente il packet filter. Per la realizzazione di tale test viene utilizzato lo strumento Iperf3 [38], il quale fornisce un software open source molto utilizzato per stabilire la banda disponibile su un collegamento della rete.

L'impatto della vNSF, oltre alle prestazioni del packet filter stesso, è misurato anche considerando il degrado di performance dovuto all'inserimento del nodo all'interno di un collegamento. Per effettuare dunque una migliore valutazione della variazione delle prestazioni date dall'inserimento della vNSF viene utilizzata preliminarmente l'architettura mostrata in Fig. 4.7, in cui viene stabilito un collegamento diretto tra le istanze VNF_A e VNF_B, su cui sono eseguiti rispettivamente il client ed il server IPerf3.

Questo approccio permette di calcolare quindi una velocità identificata come base di partenza per il test. Successivamente viene utilizzata l'architettura mostrata in Fig. 4.8, inserendo di fatto la vNSF all'interno del collegamento.

L'architettura utilizzata per l'esecuzione del test di performance viene mostrata in Fig. 4.8. Essa prevede dunque l'aggiunta della vNSF sul collegamento tra la VNF_A e la VNF_B. A questo punto si procede con il misurare l'impatto di PF al variare del numero di regole impostate utilizzando una singola connessione e 128 sessioni concorrenti. Il valore di sessioni concorrenti

Figura 4.7. Architettura per ottenere la massima velocità di collegamento

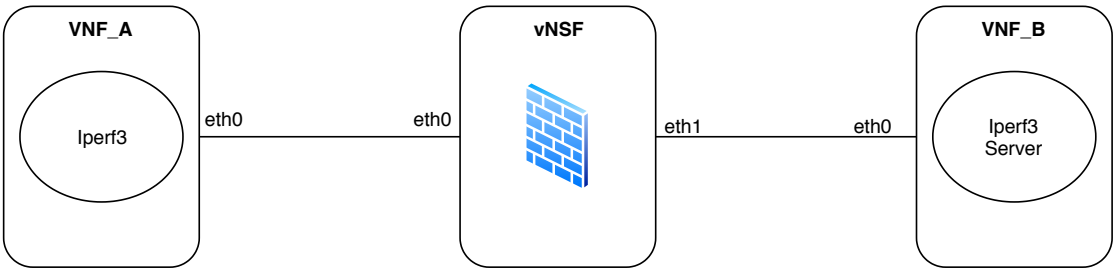

Figura 4.8. Architettura utilizzata nel test di Performance

testate risulta limitato dalle risorse di test utilizzate. Anche in questo caso il test viene eseguito tre volte, mostrando come risultato la mediana dei test. La configurazione del packet filter, per motivi di semplicità viene eseguita utilizzando direttamente la console messa a disposizione da OpenStack per gestire la vNSF. La configurazione delle macchine risulta analoga a quanto mostrato per il test funzionale e viene presentata nella Tabella 4.1.

La misura di base individuata sul collegamento è risultata essere di 1.2 Gbit/s, per entrambi i casi testati. Il risultato dei test viene mostrato in Fig. 4.9 per il caso con singola connessione, ed in Fig. 4.10 per il caso con 128 sessioni concorrenti. In entrambi i casi viene mostrato il valore di banda espresso in Mbit/s, al variare del numero di regole impostate. Nel caso di test a 128 sessioni viene mostrato il valore rappresentate la somma della banda di tutte le connessioni. Si vanno inoltre a misurare gli andamenti ottenuti per un tipo di filtraggio stateless, senza sfruttare la tabella degli stati di PF, e l'andamento ottenuto utilizzando un filtraggio stateful.

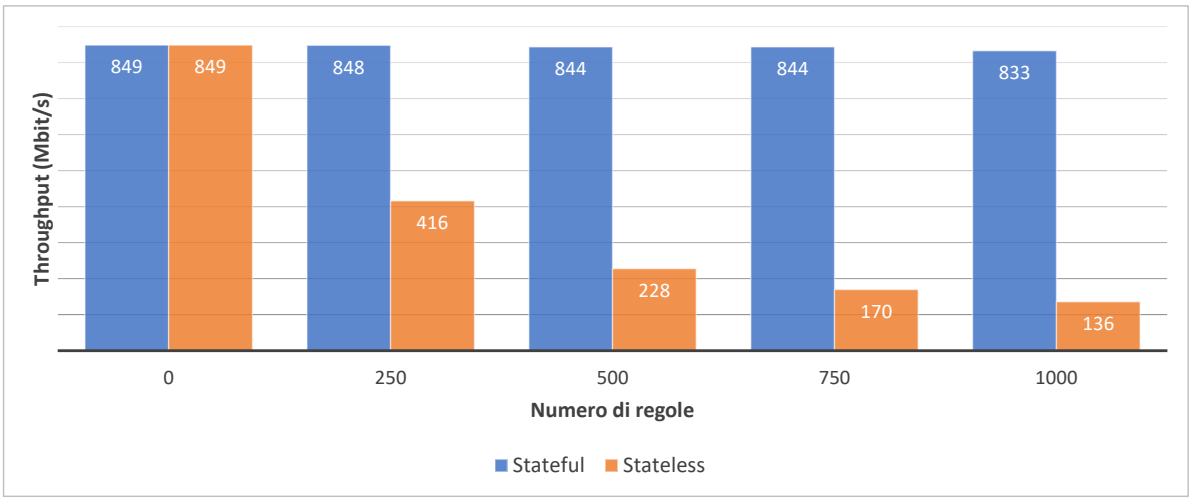

Figura 4.9. Test di performance con singola connessione

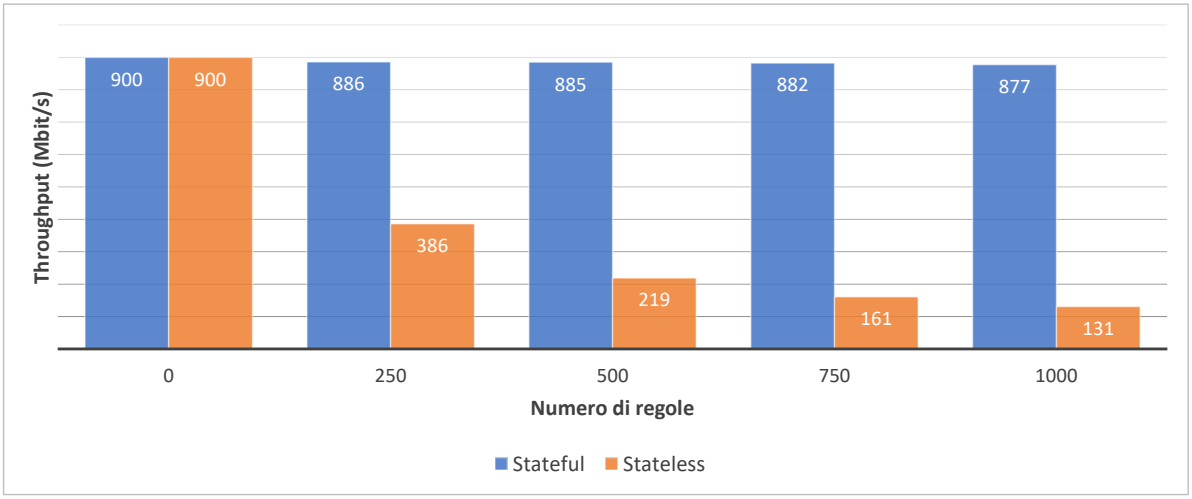

Figura 4.10. Test di performance con 128 connessioni concorrenti

Il test mostra nel caso con 0 regole il degrado di performance dovuto al solo inserimento del nodo contenente la vNSF. Considerando che ciò introduce una nuova istanza che deve effettuare il forwarding dei pacchetti con hardware non specializzato e con risorse limitate l'impatto che si verifica dall'inserimento del nuovo nodo viene considerato accettabile. Il test risulta successivamente in linea con quanto aspettato. Il degrado di performance in termini di banda, nel caso stateless, risulta essere lineare con il numero di regole da confrontare. Nel caso stateful invece, viene effettuato il confronto con le regole impostate solo per il primo pacchetto. In seguito al primo confronto, viene aggiunto un record per la connessione stabilita all'interno della tabella degli stati gestita da PF. Ciò permette di avere un impatto sulle performance quasi nullo rispetto al caso con 0 regole.

L'andamento risulta pressochè identico anche nel caso di 128 connessioni concorrenti. In tal caso, il numero di record presenti all'interno della tabella degli stati è maggiore. Tuttavia PF utilizza un albero AVL [39](dal nome degli inventori Adelson-Velskij e Landis), il quale riesce a garantire tempi di ricerca con complessità dell'ordine $\mathcal{O}(\log m)$, con m che identifica il numero di stati, anche nel caso peggiore. Questo permette dunque di gestire facilmente il numero di connessioni impostate. Come anticipato, 128 risulta il limite imposto dalle risorse utilizzate per il test.

Tuttavia il test fornisce uno spunto interessante in quanto come anticipato durante la descrizione di PF, esso risulta facilmente integrabile con lo strumento `CARP`, il quale permette di creare agevolmente dei gruppi di istanze di PF per le quali viene effettuato load balancing. Unendo dunque le buone prestazioni garantite dalla tecnologia PF nella gestione degli stati ed una buona distribuzione del carico di lavoro risulterebbe possibile ottenere un packet filter in grado di gestire il traffico introducendo riduzioni di performance molto basse. Inoltre, risulta interessante osservare come PF riesca a gestire nella tabella degli stati anche UDP ed ICMP [40], i quali sono protocolli non orientati alla connessione:

- **UDP:** quando viene inviato un pacchetto UDP, PF crea un record all'interno della tabella degli stati con un timeout molto basso. Al termine del timeout il record viene rimosso. Ciò accade nel caso in cui il pacchetto UDP rappresenti una comunicazione mono-direzionale. Nel caso in cui venga rilevata una risposta all'endpoint che ha inviato il primo pacchetto UDP viene incrementato il timeout del record creato e si gestisce la comunicazione UDP come pseudo-connessione.

- **ICMP:** le richieste e le risposte ICMP vengono gestite in modo simile alle comunicazioni UDP. Se si tratta di un messaggio di errore ICMP relativo ad una connessione già presente nella tabella degli stati, esso passa senza effettuare ulteriori operazioni di confronto.

Ciò permette di gestire tramite state table anche protocolli che in linea teorica non presentano il concetto di connessione. L'impatto maggiore viene dunque portato dall'introduzione stessa del nodo all'interno del collegamento. Impatto che risulta però indipendente dalla tecnologia di packet filtering adottata ed imposto invece dalle prestazioni di forwading di FreeBSD nell'architettura di virtualizzazione utilizzata e dalle risorse allocate. Risulterebbe dunque interessante valutare le prestazioni dell'istanza virtuale con tecnologie di virtualizzazione diverse da KVM, per valutare eventuali miglioramenti nelle prestazioni di inoltro dei pacchetti di FreeBSD.

Oltre alla misurazione dell'impatto della vNSF all'interno dell'architettura di rete utilizzata, sono state effettuate misure sui tempi necessari alla configurazione del packet filter al variare del numero di politiche inviate in una singola richiesta. Per l'esecuzione di tale test sono state inviate richieste al servizio web atto alla gestione della configurazione utilizzando il software `curl`. Tali richieste vengono inviate dalla macchina su cui OSM è in esecuzione. I risultati sono mostrati in Tabella 4.2.

Numero di regole	**Tempo di configurazione** (s)
1	0.026
250	0.277
500	0.669
750	1.065
1000	1.604

Tabella 4.2. Tempo di configurazione delle regole del packet filter

I risultati ottenuti evidenziano tempi di risposta ragionevoli per la configurazione delle policy di sicurezza. Il test mostra inoltre come sia conveniente effettuare la configurazione di più policy in una singola richiesta per ottenere un miglioramento dei tempi di risposta in proporzione al numero di regole da aggiungere. Tale miglioramento risulta ancor di maggior impatto se si considera che strumenti quali Juju richiedono tempi aggiuntivi per preparare ed eseguire la richiesta, incrementando in modo sensibile i tempi necessari. Questo rende ancor più conveniente l'aggiunta di più politiche in una singola richiesta, in modo che tali tempi aggiuntivi siano considerati una sola volta.

Capitolo 5

Conclusioni

Nel lavoro di tesi svolto si è partiti da una situazione in cui la gestione delle risorse, con l'aumentare dei dispositivi connessi alla rete, rappresenta una vera e propria criticità. Con il supporto di alcuni dati si è dimostrato che il problema colpisce anche il campo della sicurezza informatica. Come evidenziato, gli sviluppi portati in ambito tecnologico hanno fornito ai cyber criminali maggiori risorse per effettuare attacchi che portano a perdite economiche e diffusione di dati sensibili. Ciò ha dunque evidenziato il bisogno di un nuovo approccio al design della rete che sia in grado di fornire una maggiore dinamicità nella distribuzione delle soluzioni, per permettere un adattamento rapido alle situazioni di pericolo presenti. Come soluzione per la situazione descritta è stato individuato un nuovo metodo di progettazione della rete che ha portato all'introduzione dei concetti di Cloud Computing, NFV e SDN. Come visto tali paradigmi hanno introdotto notevoli cambiamenti nel modo di concepire i servizi di rete che hanno portato ad una visione sempre più orientata al software. I concetti di Cloud Computing ed SDN sono stati introdotti in quanto tecnologie abilitanti per l'utilizzo di una architettura di rete distribuita e softwarizzata.

Maggiore attenzione è stata posta sul paradigma NFV. Nel corso del lavoro di tesi è stato affrontato uno studio sui blocchi previsti per permettere la virtualizzazione dei servizi di rete e per abilitare l'utilizzo delle VNF. Attraverso la definizione di VNF e lo studio dei moduli e delle interfacce da implementare affinchè la funzione virtualizzata sia integrabile in tale contesto di rete si è giunti alla definizione di vNSF come funzione di rete virtuale specializzata nell'ambito della sicurezza informatica.

Come obiettivo di tesi si è voluto aggiungere un risvolto pratico alla fase di studio del concetto vNSF. A tal proposito è stata introdotta la funzionalità che si intendeva implementare, individuata nel packet filter. Come lavoro preliminare sono state affrontate e confrontate due tecnologie di packet filtering ampiamente utilizzate come Netfilter/Iptables e PF. Questo ha permesso di conoscere, oltre alle tecnologie messe a confronto, anche strumenti di benchmarking per valutare le prestazioni di un prodotto di questo tipo. In aggiunta allo studio della tecnologia utilizzata per la realizzazione della vNSF si è anche voluta dare un'introduzione del metodo di configurazione applicato. È stato quindi introdotto il concetto di gestione policy-based come metodo per definire le regole da applicare ai diversi moduli della rete atti a garantire un funzionamento corretto e sicuro. Come punto di partenza per il lavoro realizzato, è stata analizzata l'implementazione effettiva del modulo di gestione policy-based e di come una vNSF interagisca con esso all'interno del progetto Europeo SHIELD. Questo ha permesso di definire i principali blocchi da realizzare per integrare il packet filter in un simile contesto.

Durante la fase di implementazione della vNSF si ha avuto la possibilità di approfondire strumenti open source atti alla realizzazione di un contesto cloud. Tra questi, è stato individuato in Open Source MANO uno strumento per la gestione e l'orchestrazione delle VNF ed in OpenStack uno strumento in grado di fornire risorse su cui eseguire in modo concreto le istanze delle funzioni virtuali. Questa fase del lavoro ha quindi permesso un interessante approfondimento delle tecnologie utilizzate per la realizzazione di un contesto NFV. Oltre a strumenti di infrastruttura, durante la fase di implementazione sono state affrontate le metodologie di configurazione di una VNF. Ciò ha permesso di toccare con mano strumenti come Ansible e Juju per la gestione del ciclo di vita di un'istanza cloud.

Il test finale ha messo in evidenza come l'obiettivo di ottenere una vNSF in grado di essere gestita utilizzando un approccio PBM in ambito NFV sia stato raggiunto tramite l'implementazione realizzata. Al termine del lavoro si è infatti in grado di istanziare la funzione di sicurezza virtuale utilizzando gli strumenti sopracitati e di configurare il filtraggio dei pacchetti tramite la specifica di politiche con un livello medio di astrazione. Il risultato ottenuto si è dimostrato in linea con quanto aspettato, mostrando come la tecnologia PF sia in grado di gestire bene il filtraggio dei pacchetti utilizzando un approccio stateful, mostrando invece un decremento di performance lineare con il numero di regole nel caso di filtraggio stateless. I tempi di configurazione sono risultati anch'essi accettabili, mostrando tempistiche relativamente brevi per l'applicazione di nuove regole, soprattutto includendo più politiche in una singola richiesta. La criticità maggiore si è rivelata nelle capacità di forwarding dei pacchetti che l'istanza di FreeBSD ha dimostrato sulla tecnologia di virtualizzazione KVM.

5.1 Sviluppi futuri

Quanto realizzato rappresenta un singolo blocco di virtualizzazione che può essere integrato in un contesto più esteso. Si può infatti inglobare la funzionalità di packet filtering in un contesto di orchestrazione in cui si vanno a formare catene di servizi complessi, approfondendo concetti relativi alla connessione delle funzioni di rete virtualizzate che permettano di applicare nozioni proprie del paradigma SDN. Ciò permetterebbe di verificare il funzionamento di tale blocco in un contesto di applicazione reale, connettendo più VNF atte a fornire un servizio di rete complesso e verificare le prestazioni della soluzione realizzata. Questo permetterebbe inoltre di affiancare funzioni di sicurezza complementari al packet filter che siano in grado di garantire un livello di sicurezza più alto. Sono state infatti introdotte vulnerabilità note del packet filter, che quindi non fornisce una soluzione di sicurezza completa ed a se stante. Durante la fase di test sono inoltre state iniettate manualmente le politiche in formato MSPL, senza passare dalla definizione della politica ad alto livello. Sarebbe dunque interessante integrare la soluzione realizzata con un modulo in grado di effettuare la gestione delle policy, in modo da automatizzare il processo di configurazione delle regole.

Tra le criticità presentate durante il percorso di tesi, è risultato che le prestazioni di FreeBSD come router di pacchetti in un ambito virtualizzato con tecnologia KVM introducano un calo di performance della vNSF non dipendente dal filtraggio stesso dei pacchetti. Risulterebbe dunque utile effettuare ulteriori ricerche su tale argomento, sperimentando come variano le prestazioni di base di FreeBSD a livello di networking utilizzando diverse tecnologie di virtualizzazione (e.g. Docker, Xen).

Tra i lavori futuri è inoltre possibile estendere la gestione di PF. Attualmente, il formato MSPL utilizzato per la configurazione del packet filter garantisce le informazioni utili per impostare regole di base per il filtraggio dei pacchetti. È possibile estendere il set di parametri gestibile tramite la definizione di una policy MSPL in modo da sfruttare al meglio le potenzialità che PF mette a disposizione dell'utente.

Appendice A

Manuale utente

In questa appendice viene mostrato come è possibile utilizzare la soluzione realizzata, mostrando in che modo navigare l'interfaccia di Open Source MANO per poter istanziare il NS ed in che modo configurare la VNF.

A.1 Utilizzo del NS tramite interfaccia

Per utilizzare il NS realizzato in questa tesi si può usare l'interfaccia messa a disposizione da OSM. Dopo aver effettuato l'installazione e l'avvio del servizio OSM, esso risulterà raggiungibile tramite browser alla porta 80. Sarà dunque sufficiente effettuare l'accesso all'indirizzo `http://OSM_ADDRESS`, ed autenticarsi utilizzando le credenziali di default (nome utente: admin, password: admin) o le credenziali impostate.

In Fig. A.3 viene mostrata la pagina home di OSM. In tale pagina viene mostrato un resoconto dello stato attuale dell'applicazione. Poniamo particolare attenzione alle informazioni relative ai package di NS e di VNF caricate all'interno di OSM ed alle corrispettive istanze avviate. Tali informazioni sono raggiungibili anche attraverso il menu di navigazione laterale messo a disposizione dall'interfaccia grafica. Tramite tale menu è possibile accedere, oltre che alle sezioni evidenziate nell'overview iniziale, ad altre informazioni utili. È ad esempio possibile configurare manualmente gli account VIM utilizzati da OSM selezionando l'opzione "VIM Accounts". In questa guida all'utilizzo della vNSF si parte con un VIM configurato ed i package di NS e VNF pre-caricati su OSM. Per informazioni riguardanti i passaggi preliminari si rimanda alla guida di OSM raggiungibile a [41].

Per poter proseguire con l'istanziazione del NS è necessario accedere alla sezione apposita utilizzando il collegamento alla lista di NS messo a disposizione nella pagina home di OSM, oppure utilizzando la sezione "NS Packages" del menu laterale. In tale sezione verrà mostrata la lista di NS istanziabili, in particolare il record di interesse per il NS realizzato viene mostrato in Fig. A.1. Il NS realizza quanto descritto nei capitoli precedenti, istanziando tre VNF, due

Short Name	Identifier	Description	Vendor	Version	Actions
TestNS_nsd	cf30980b-b44d-4c74-ab48-d307515d01cb	Generated by OSM package generator	Antonio Checola	1.0	

Figura A.1. NS di test in OSM

utilizzate per il test ed una contenente PF. È possibile accedere alle informazioni relative a tali VNF accedendo alla sezione "VNF Packages". Al momento dell'istanziazione del NS è necessario specificare per l'istanza:

- nome;
- descrizione;
- id del NS di riferimento;
- VIM utilizzato;

Durante la fase di istanziazione è possibile tenere traccia dei progressi del NS e delle VNF incluse in esso. Per tener traccia dei progressi del NS è necessario recarsi nella sezione "NS Instances", in cui verrà mostrata l'interfaccia in Fig. A.4. Grazie alle informazioni contenute in tale sezione è possibile tener traccia dello stato di avanzamento del NS. Tale fase comprende l'allocazione delle risorse sul VIM, la creazione del proxy charm per la gestione della configurazione tramite Juju e l'esecuzione delle operazioni di inizializzazione. In tale fase dunque viene anche eseguito il playbook Ansible definito per la configurazione della vNSF. Per poter effettuare le operazioni elencate questa fase richiede qualche minuto.

Una volta terminate le operazioni, il NS risulta nello stato "attivo", a questo punto è possibile effettuare le operazioni di configurazione sulla vNSF. Sia durante la fase di istanziazione, sia dopo che essa giunge al termine, è possibile monitorare le istanze utilizzando l'interfaccia messa a disposizione da OpenStack. Tale interfaccia risulterà raggiungibile, come per OSM, utilizzando un browser ed accedendo all'indirizzo `http://OpenStack_Address`. Essa viene mostrata in Fig. A.5 e permette di ottenere informazioni relative allo stato delle istanze create ed agli indirizzi assegnati. Inoltre tramite l'utilizzo di tale interfaccia è possibile accedere ai log delle operazioni eseguite ed utilizzare console messe a disposizione in modo da poter eseguire operazioni direttamente sull'istanza creata, senza utilizzare strumenti esterni (e.g. Juju). Questo è utile per scopi di debug, in quanto il contesto descritto nella tesi non prevede l'utilizzo di configurazioni effettuate con tale approccio.

A.2 Configurazione della vNSF con Juju

La configurazione della vNSF può essere effettuata tramite i comandi messi a disposizione da Juju. Al fine di poter effettuare la configurazione è necessario ritrovare le informazioni relative al proxy charm utilizzato per gestire la vNSF istanziata. Accedendo alla macchina su cui viene eseguito OSM è possibile accedere a tali informazioni utilizzando il comando

```
$ juju status
```

l'esecuzione di tale istruzione produrrà un output simile a quanto mostrato in Fig. A.2.

```
Model    Controller  Cloud/Region         Version  SLA          Timestamp
default  osm         localhost/localhost  2.6.10   unsupported  19:51:30Z

App             Version  Status  Scale  Charm          Store  Rev  OS      Notes
vnsf-test-c-aa           active      1  ansible-charm  local    7  ubuntu

Unit               Workload  Agent  Machine  Public address  Ports  Message
vnsf-test-c-aa/0*  active    idle   7        10.91.240.74           Ready!

Machine  State    DNS           Inst id        Series  AZ  Message
7        started  10.91.240.74  juju-ddff3d-7  xenial      Running
```

Figura A.2. Esempio di stato di un proxy charm

da quanto mostrato è possibile avere informazioni sui nomi assegnati da Juju per app e unit create, nell'esempio in analisi individuati rispettivamente in *vnsf-test-c-aa* ed *vnsf-test-c-aa/0*. Viene inoltre mostrato lo stato corrente del charm, utile per verificare le operazioni in esecuzione o se il charm è in attesa di altri comandi.

Risulta a questo punto possibile avere visione delle operazioni invocabili da tale charm sulla vNSF gestita, specificando il comando:

```
$ juju list-actions APP_NAME
```

sostituendo APP_NAME con il nome dell'app istanziata per la propria VNF.

A questo punto risulta possibile eseguire le operazioni elencate utilizzando il comando:

```
$ juju run-action UNIT_NAME ACTION_NAME [PARAMS]
```

utilizzando i valori opportuni per il nome della unit, il nome della action da eseguire e specificando eventuali parametri nel formato *key=value*. Per poter impostare nuove policy si può utilizzare, a titolo di esempio, l'istruzione:

```
$ juju run-action UNIT_NAME set-policies policies='MSPL_policy'
```

specificando le politiche direttamente in formato MSPL, per il quale un esempio viene mostrato in 4.2.1. L'esecuzione del comando restituirà un identificativo utilizzabile per seguire lo stato della richiesta e per ottenerne l'output. Nel caso in cui si sia interessati al solo esito della action è possibile eseguire il comando

```
$ juju show-action-status <ID>
```

se invece si è interessati anche all'output prodotto dalla action, come può accadere per l'action che restituisce la lista di regole impostate sul packet filter, si può utilizzare il comando

```
$ juju show-action-output <ID>
```

è possibile infine tenere traccia delle operazioni eseguite da Juju utilizzando l'output di debug messo a disposizione dal comando

```
$ juju debug-log
```

Per la vNSF le azione possibili sono state descritte nel capitolo di implementazione. Tramite l'utilizzo di queste operazioni, utilizzando i comandi descritti, risulta possibile configurare la funzione virtualizzata di filtraggio dei pacchetti realizzata nel contesto del lavoro di tesi. Si propone a scopo riassuntivo la Tabella A.1 ,per un riscontro immediato sulle azioni a disposizione.

Nome	**Parametri**	**Descrizione**
get-policies	-	restituisce le politiche impostate sulla vNSF
set-policies	policies:STRING	imposta le politiche MSPL specificate come parametro
delete-policies	-	elimina tutte le politiche impostate sulla vNSF
delete-policy	policy:INT	elimina la politica identificata dal valore del parametro
get-quick	-	restituisce lo stato di default sull'utilizzo di quick
set-quick	quick:BOOL	imposta lo stato di default sull'utilizzo di quick

Tabella A.1. Tabella riassuntiva delle azioni di gestione delle politiche

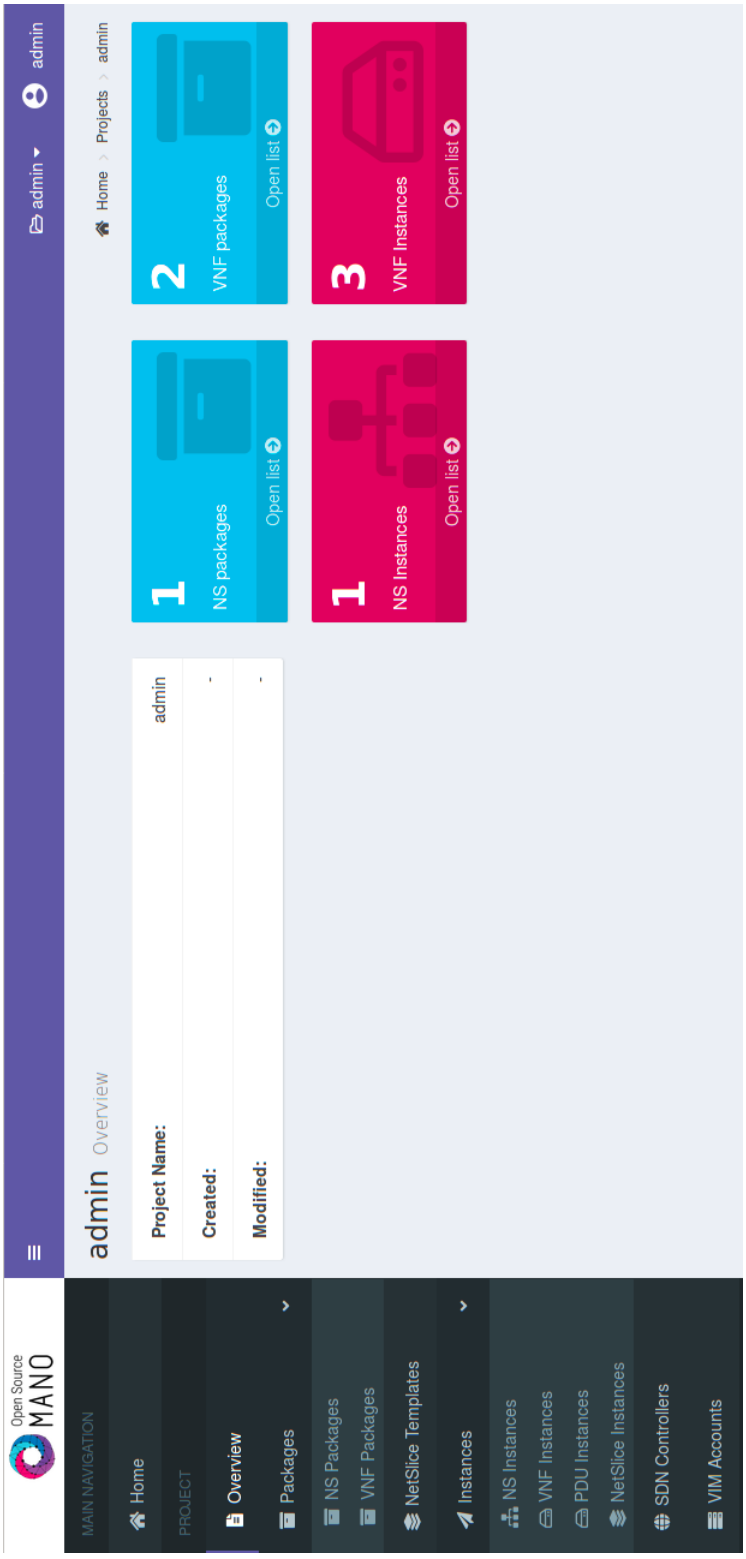

Figura A.3. Home page di Open Source MANO Release 5

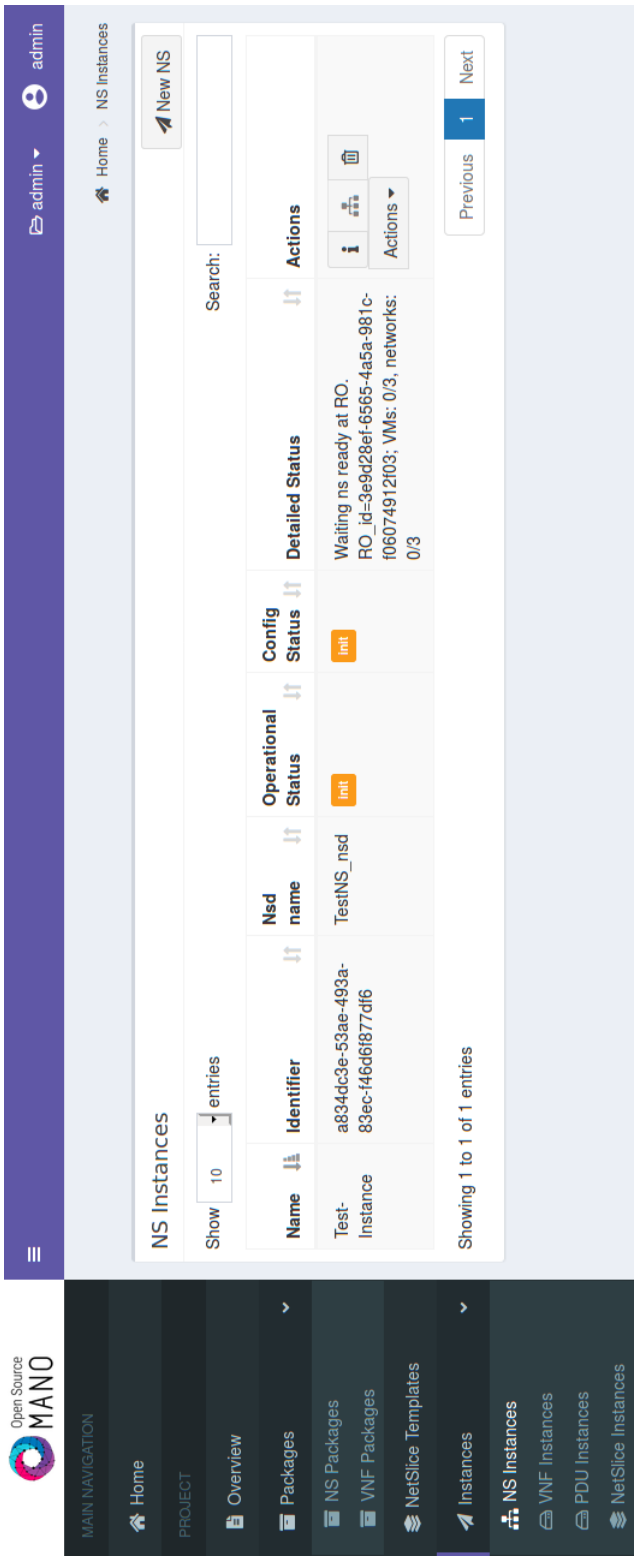

Figura A.4. Interfaccia di istanziazione del NS in OSM

Figura A.5. Interfaccia di OpenStack

Appendice B

Manuale dello sviluppatore

In questo manuale viene affrontato in dettaglio il codice sviluppato per descrivere in che modo esso possa essere esteso in applicazioni future. Viene inoltre fornita una spiegazione su come riprodurre l'ambiente di sviluppo utilizzato nel contesto della realizzazione della vNSF.

B.1 Installazione di Open Source MANO Release FIVE

Per poter usufruire della soluzione realizzata è necessario installare su una macchina fisica o su VM lo strumento OSM. Per la release FIVE è importante che la macchina su cui viene installato OSM presenti come sistema operativo Ubuntu 16.04 nella variante a 64-bit, in quanto questa rappresenta la scelta preferibile per l'installazione di OSM. Inoltre sono richieste le seguenti risorse:

- **Risorse minime:** 2 CPU, 4 GB RAM, 20 GB di disco ed un'interfaccia con accesso ad internet.
- **Risorse raccomandate:** 2 CPU, 8 GB RAM, 40 GB di disco ed un'interfaccia con accesso ad internet.

Con tali risorse a disposizione si può procedere all'installazione di OSM release FIVE. A tale scopo si procede con l'ottenere il file di installazione e renderlo eseguibile tramite i comandi

```
$ wget https://osm-download.etsi.org/ftp/osm-5.0-five/install_osm.sh
$ chmod +x install_osm.sh
```

A questo punto si può procedere all'installazione di OSM utilizzando il comando

```
$ ./install_osm.sh -t v5.0.5
```

Risulta tuttavia utile per la risoluzione di eventuali problemi di installazione re-indirizzare l'output di tale processo su un file di log. Ciò è possibile utilizzando, in alternativa all'opzione precedente, il comando

```
$ ./install_osm.sh -t v5.0.5 2>&1 | tee osm_install_log.txt
```

Durante l'esecuzione del file install_osm.h viene gestita anche l'installazione di tutte le dipendenze utilizzate, quali LXD, juju, docker. Per tali dipendenze bisogna dunque selezionare l'opzione "y" quando richiesto se procedere con l'installazione di tali strumenti. Verranno inoltre visualizzati alcuni messaggi per la configurazione di LXD ai quali bisogna rispondere nel seguente modo:

```
Do you want to configure the LXD bridge? Yes
Do you want to setup an IPv4 subnet? Yes
<< Utilizzare il valore di default per le opzioni proposte>>
Do you want to setup an IPv6 subnet? No
```

Al termine del processo OSM release FIVE, risulterà installato. Per l'utilizzo dell'interfaccia fornita da OSM si faccia riferimento al manuale utente (Appendice A). Per installazioni alternative che prevedano l'utilizzo di componenti aggiuntivi (e.g. VIM Emulator, ELK) si faccia riferimento alla guida di OSM Release FIVE [41].

B.2 Installazione di OpenStack

In questa sezione viene affrontata l'installazione di OpenStack, utilizzato come VIM in OSM. L'opzione di installazione scelta è quella fornita da DevStack, ossia una soluzione messa a disposizione degli sviluppatori di OpenStack che permette di avere una versione funzionante di questo strumento e con i servizi base installati con poche operazioni.

Anche nel caso di OpenStack, il sistema operativo della macchina su cui si desidera eseguire l'installazione deve essere Ubuntu 16.04 LTS. Le operazioni preliminari da eseguire consistono nel creare un utente devstack e creare una copia locale del repository contenente il file di installazione con i seguenti comandi

```
$ sudo useradd -s /bin/bash -d /opt/stack -m stack
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
$ sudo su - stack
$ git clone https://git.openstack.org/openstack-dev/devstack
$ cd devstack
```

A questo punto si può procedere con l'esecuzione del file di installazione o creare un file **local.conf** in cui specificare particolari preferenze di installazione. In particolare è possibile specificare l'aggiunta di servizi oltre quelli base, precaricare immagini o fornire altre informazioni per cui viene fornito maggior dettaglio nella guida raggiungibile a https://docs.openstack.org/devstack/latest. Per procedere con l'installazione è sufficiente eseguire

```
$ ./stack.sh
```

al termine dell'esecuzione di tale script si potrà usufruire dei servizi base forniti da OpenStack e sarà possibile configurarlo per l'utilizzo con OSM.

B.3 Configurazione di OpenStack in Open Source MANO

Una volta completata l'installazione di OpenStack è possibile utilizzarlo come VIM in OSM. Risulta fondamentale che l'API fornita da OpenStack risulti raggiungibile da OSM, ed in particolare dal modulo RO, il quale come descritto nei capitoli precedenti è responsabile della gestione delle risorse.

Bisogna successivamente creare una rete di management all'interno di OpenStack con DHCP abilitato. Tale rete deve essere raggiungibile da OSM, in particolar modo dal modulo VCA (Juju). Si può ad esempio utilizzare i comandi

```
$ neutron net-create mgmt --provider:network_type=vlan
    --provider:physical_network=physnet_em1 --provider:segmentation_id=500
    --shared
$ neutron subnet-create --name subnet-mgmt mgmt 10.208.0.0/24
    --allocation-pool start=10.208.0.2,end=10.208.0.254
```

per configurare una rete che utilizzi l'interfaccia fisica em1, VLAN 500 sulla sottorete 10.208.0.0/24. Risulta a questo punto necessario registrare un utente con i diritti di creare o eliminare nuovi flavor. A tale scopo si può creare un utente con il ruolo "admin" o modificare il file */etc/nova/policy.json* per permettere tali operazioni all'utente scelto.

Per utilizzare il NS realizzato in questa tesi bisogna effettuare il caricamento dell'immagine utilizzata dalle VNF. Per poter utilizzare FreeBSD all'interno di OpenStack, è stato utilizzato bsd-cloudinit. Per poter realizzare un'immagine di FreeBSD personalizzata ed utilizzabile all'interno

del contesto di tesi si faccia riferimento alla guida al link https://docs.openstack.org/image-guide/freebsd-image.html. Facendo riferimento alla guida, l'immagine utilizzata è stata ulteriormente personalizzata creando un ulteriore utente *osm* con permessi di amministratore utilizzato per configurare la vNSF. Per caricare l'immagine si può utilizzare il comando

```
$ openstack image create --file="path/to/freebsd_image"
    --container-format=bare --disk-format=qcow2 --public FreeBSD
```

Per le VM create con tale immagine bisogna modificare il security group applicato in modo da abilitare il traffico SSH (porta TCP 22).

L'ultima operazione necessaria per utilizzare OpenStack è registrarlo come VIM all'interno di OSM. A tale scopo si utilizza il comando

```
$ osm vim-create --name openstack-site --user admin --password password
    --aut_url http://Openstack_IP/identity/v3 --tenant admin --account_type
    openstack --config='{project_domain_name: name, user_domain_name: name}'
```

sostituendo i parametri utilizzati con valori opportuni.

B.4 Estensione del modulo di configurazione

In questa sezione verrà analizzato in che modo è possibile estendere il modulo di configurazione utilizzato ed aggiungere nuove azioni al Juju charm realizzato.

B.4.1 Operazioni di configurazione nel modulo Ansible

L'inizializzazione viene eseguita, come descritto nei capitoli precedenti, tramite l'utilizzo di un playbook Ansible. Facendo riferimento al codice fornito nella directory charm è possibile accedere alle informazioni utilizzate per l'esecuzione di Ansible. La creazione dell'inventario Ansible e del file di configurazione viene effettuata con lo script **ansible-charm/reactive/ansible_charm.py**, nella seguente funzione

```
def config_changed():
    try:
        # Retrieve the ssh parameter
        cfg = config()
        # edit ansible hosts file with the VNF parameters
        h = open("/etc/ansible/hosts", "wt")
        h.write("[freebsd]\n")
        h1 = "{} ansible_connection=paramiko ansible_ssh_user={}
            ansible_ssh_pass={}
            ansible_python_interpreter=/usr/local/bin/python2.7\n".format(
            cfg['ssh-hostname'],cfg['ssh-username'],cfg['ssh-password'])
        h.write(h1)
        h.close()
        # edit ansible config to enable ssh connection with th VNF
        c = open("/etc/ansible/ansible.cfg", "wt")
        c.write("[freebsd]\n")
        c.write("host_key_checking = False\n")
        c.close()
        set_flag('ansible-charm.configured')
        status_set('active', 'Running VNF Configuration!')
    except Exception as e:
        action_fail('command failed: {}, errors: {}'.format(e, e.output))
    return
```

in questa funzione si procede con la definizione del nodo freebsd, per il quale viene utilizzato Paramiko per effettuare la connessione e per il quale viene utilizzato un interprete Python in un path specifico per il sistema operativo FreeBSD. Vengono inoltre impostate le informazioni relative a hostname, username e password da utilizzare per la connessione con l'istanza della vNSF. Tali informazioni vengono raccolte dalla configurazione utilizzata e a differenza dei parametri precedenti, che sono *hardcoded*, possono variare tra le diverse istanziazioni della vNSF a seconda della configurazione utilizzata. Viene inoltre creato un file di configurazione in cui vengono specificate alcune opzioni desiderate. Nel caso corrente viene specificato di non effettuare il controllo sull'host key. Per aggiungere nuove preferenze relative all'utilizzo di Ansible o nuove opzioni di configurazione risulta dunque necessario modificare tale funzione per prevedere l'utilizzo dei nuovi parametri desiderati o modificare quelli esistenti.

Le operazione di inizializzazione vengono definite in **ansible-charm/playbook/playbook.yaml**. In questo file si trova l'effettiva realizzazione del playbook utilizzato per inizializzare la vNSF. Tali operazioni vengono effettuate sull'host freebsd come definito in precedenza nell'inventario. Per aggiungere nuove operazioni da eseguire in fase di inizializzazione è sufficiente aggiungere un nuovo task. Un esempio di task è dato dal seguente

```
tasks:
    ...
    - name: check out web service repo
      git:
        repo: 'https://REPOSITORY_ADDRESS/pf_webservice.git'
        dest: /home/osm/pf_webservice
    ...
```

con il seguente task viene clonato il repository nel quale è contenuto il web service utilizzato dalla vNSF per la gestione delle richieste. I task definiti nel playbook vengono eseguiti in modo sequenziale, bisogna dunque prestare attenzione all'ordine in cui si vuole aggiungere nuove operazioni di inizializzazione. Risulta tuttavia possibile assegnare dei tag ai task definiti, in modo da invocare il singolo task di configurazione. Un esempio è dato da

```
tasks:
    ...
    - name: stop the web service
      shell: kill $( lsof -i:{{ api_port }} -t )
      tags:
        - stopService
    ...
```

i tag nell'implementazione corrente vengono utilizzati per effettuare operazioni di gestione dello stato della vNSF eseguibili durante il lifecycle. Tale operazione, al quale è stato assegnato il tag "stopService", è ad esempio eseguita quando viene invocata l'operazione "stop" messa a disposizione dal Juju charm. Il codice per eseguire tali operazioni è definito in **ansible-charm/reactive/ansible_charm.py**, come nel caso della definizione dell'inventario. La funzione "stop" è definita nel seguente modo

```
def stop():
    try:
        #Stop the web service
        path = find('playbook.yaml','/var/lib/juju/agents/')
        call = ['ansible-playbook', path, '--extra-vars', "api_port=" +
            cfg['rest-api-port'], '--tags', "resetFile,resetPF,stopService"]
        subprocess.check_call(call)
        status_set('active', 'Ready!')
    except Exception as e:
        action_fail('command failed: {}, errors: {}'.format(e, e.output))
    finally:
        remove_flag('actions.stop')
```

risulta dunque possibile aggiungere al playbook nuove operazioni che verranno eseguite durante l'inizializzazione della vNSF. Risulta inoltre possibile modificare le operazioni già definite. Un esempio è dato dal caso in cui si voglia utilizzare un nuovo web service definito in un nuovo repository. In tal caso sarà sufficiente modificare il repository sorgente e definire le operazione per l'avvio del nuovo web service. Risulta inoltre possibile creare operazioni di gestione della vNSF invocabili direttamente dal Juju charm durante il lifecycle dell'istanza della funzione virtuale.

B.4.2 Aggiunta di azioni al Juju charm

L'API realizzata per il Juju charm è stata analizzata nel capitolo riguardante l'implementazione della vNSF. È possibile estendere tale API aggiungendo nuove azioni in grado di implementare nuove funzionalità per la vNSF.

Per fornire un esempio sul come aggiungere una nuova azione al charm viene presentato in che modo è stata aggiunta la funzione **delete-policy**. Rispetto ai sorgenti allegati alla tesi si faccia riferimento al codice contenuto nella cartella charm. Come primo passo è stato aggiunto il file **ansible-charm/actions/delete-policy** con il contenuto

```
#!/usr/bin/env python3
import sys
sys.path.append('lib')

from charms.reactive import main
from charms.reactive import set_state
from charmhelpers.core.hookenv import action_fail, action_name

set_state('actions.{}'.format(action_name()))

try:
    main()
except Exception as e:
    action_fail(repr(e))
```

questo file viene utilizzato per impostare lo stato, in modo che il Juju charm invochi il codice definito per gestire l'azione che si desidera eseguire. È fondamentale che tale file risulti eseguibile, a tale scopo bisogna eseguire il comando

```
$ chmod +x ansible-charm/actions/delete-policy
```

Nell'implementazione attuale, dopo che lo stato è impostato alla action invocata, viene eseguita la funzione di gestione implementata nello script **ansible-charm/reactive/ansible_charm.py**. La funzione viene identificata con l'annotazione `@when(...)` la quale permette di specificare clausole di esecuzione del codice riferito da tale annotazione. Nell'esempio preso in considerazione, per la gestione di cancellazione della singola policy viene definita la seguente funzione

```
@when("ansible-charm.configured")
@when("actions.delete-policy")
def delete_policy():
    policy = action_get("policy")
    args = [("actions.delete-policy", "/flushRule/v1/{}".format(policy),
    "DELETE")]
    ssh_curl_call(args)
```

ciò permette di realizzare il paradigma *reactive* descritto nei capitoli precedenti. In particolare, la funzione mostrata viene eseguita all'invocazione dell'azione utilizzata come esempio e nel momento in cui il charm risulta nello stato "configured".

L'ultimo passo per aggiungere una nuova azione al charm è quello di dichiarare tale azione e definirne i parametri utilizzati. Come visto nella funzione utilizzata per gestire delete-policy, viene utilizzato il parametro policy. È dunque importante definire quali sono i parametri utilizzati,

stabilendo per essi eventuali valori di default e definendo se tali parametri risultano necessari o opzionali nell'invocazione dell'azione. Per dichiarare una nuova azione bisogna modificare il file **ansible-charm/layer.yaml** aggiungendo le informazioni necessarie. Nel caso d'esempio è possibile individuare la dichiarazione

```
...
delete-policy:
        description: "Delete a specific MSPL policy."
        params:
            policy:
                description: "Policy ID."
                type: integer
                default: 1
            required:
                - policy
...
```

al termine di tali operazioni, l'azione risulta integrata nel charm ma non ancora utilizzabile dalla vNSF. Si prosegue con una descrizione sulla compilazione del charm e su come tale modifica venga integrata sul lato della vNSF.

B.4.3 Configurazione dell'ambiente Juju

Le operazione elencate, affinchè possano essere effettivamente applicate alla vNSF richiedono che venga effettuata la *build* del charm. Affinchè ciò sia possibile bisogna configurare l'ambiente Juju nel modo descritto in questa sezione.

L'installazione di Juju richiede l'utilizzo del servizio *snapd*. Su sistema operativo Ubuntu è possibile installare tale servizio con i comandi

```
$ sudo apt update
$ sudo apt install snapd
```

nel caso di utilizzo di un sistema operativo differente si rimanda alla guida di installazione raggiungibile a https://snapcraft.io/docs/installing-snapd.

Si può successivamente procedere all'installazione di Juju tramite il servizio installato utilizzando il comando

```
$ snap install charm
```

Per effettuare la configurazione del workspace si può procedere con l'esecuzione delle istruzioni

```
$ mkdir -p $HOME/charms/layers
$ export JUJU_REPOSITORY=$HOME/charms
$ export LAYER_PATH=$JUJU_REPOSITORY/layers
```

Con tali operazioni il workspace per Juju risulta configurato ed è possibile effettuare la build del charm in modo da poter utilizzare le estensioni apportate. A tale scopo bisogna spostare la cartella **ansible-charm** all'interno del LAYER_PATH. Per poter effettuare la build bisogna spostarsi nella cartella contenente il charm e successivamente effettuare la build. Per completare tali operazioni, dopo essersi accertati che si è connessi alla rete, si può eseguire

```
$ cd $LAYER_PATH/ansible-charm
$ charm build
```

il charm compilato, il quale è collocato nella cartella **$HOME/charms/builds**, può essere integrato all'interno della vNSF. Con riferimento al codice consegnato, tale charm dovrà essere integrato con la vNSF copiandolo nella cartella **VNF/PFl3Filter_vnfd/charms**.

B.4.4 Integrazione della modifica nella vNSF

Le modifiche apportate richiedono, nel caso di aggiunta di funzionalità al charm, l'adattamento del descrittore della vNSF. Le azioni del charm devono essere mappate all'interno del descrittore tra le primitive di configurazione disponibili. Facendo riferimento al codice allegato, il descriptor della vNSF è dato dal file **VNF/PFl3Filter_vnfd/PFl3Filter_vnfd.yaml**. La sezione relativa al charm è rappresentata da

```
...
        vnf-configuration:
            juju:
                charm: ansible-charm
            config-primitive:
            - name: config
                parameter:
                - data-type: STRING
                    default-value: <rw_mgmt_ip>
                    name: ssh-hostname
                - data-type: STRING
                    default-value: osm
                    name: ssh-username
                - data-type: STRING
                    default-value: osm4u
                    name: ssh-password
            - name: ansible-playbook
            - name: start
            - name: stop
            - name: restart
            - name: get-quick
            - name: set-quick
                parameter:
                - data-type: BOOLEAN
                    default-value: True
                    name: quick
            - name: set-policies
                parameter:
                - data-type: STRING
                  default-value: ""
                  name: policies
            - name: get-policies
            - name: delete-policy
                parameter:
                - data-type: INTEGER
                  default-value: 1
                  name: policy
            - name: delete-policies
        initial-config-primitive:
        - seq: '1'
          name: config
          parameter:
            - name: ssh-hostname
              value: <rw_mgmt_ip>
            - name: ssh-username
              value: osm
            - name: ssh-password
              value: osm4u
        - seq: '2'
          name: ansible-playbook
```

la parte di descrittore mostrata viene utilizzata per definire le azioni di configurazione esposte dal Juju charm, definendo per ognuna di esse i parametri utilizzati con dei valori di default. Vengono inoltre specificate le primitive che devono essere eseguite per effettuare la configurazione day-1. Esse vengono dichiarate nella sezione denominata "initial-config-primitive". Nel caso in cui vengano effettuati cambiamenti che vadano a modificare l'interfaccia del charm, la sezione di descrittore mostrata deve essere adattata alle nuove azioni esposte.

Affinchè le modifiche apportate alla vNSF siano utilizzabili all'interno dell'ambiente di testing è richiesto che il package contenente la vNSF venga ricreato. Per poter creare il package della vNSF si può clonare il repository devops per sviluppatori di OSM eseguendo

```
$ git clone https://osm.etsi.org/gerrit/osm/devops
```

in questo repository viene fornito lo script per generare il .tar.gz con il quale è possibile effettuare l'on-board della vNSF. Sarà a questo punto sufficiente eseguire lo script sulla vNSF invocando

```
$ ./devops/descriptor-packages/tools/generate_descriptor_pkg.sh -t vnfd -N
    VNF/PFl3Filter_vnfd
```

B.5 Estensione del servizio di gestione

Come descritto nel corso della tesi, le richieste di configurazione di PF vengono gestite tramite l'utilizzo di un web service realizzato in Python e sfruttando il framework Flask. Pur avendo fornito l'implementazione del servizio di gestione in allegato con la tesi, esso viene clonato durante la fase di inizializzazione direttamente da un repository pubblico. Per modificare tale modulo è dunque necessario apportare modifiche alla versione remota utilizzata attualmente. Si può alternativamente, come descritto in precedenza, modificare la sorgente da cui creare una copia del web service modificando il playbook Ansible di inizializzazione e fornendo il link ad un repository alternativo. Vengono ora affrontate le sezioni in cui il codice è diviso al fine di facilitare la modifica del servizio web per inglobare nuove funzionalità.

Il web service espone l'API affrontata nel capitolo 4. È possibile estendere il comportamento del servizio aggiungendo la gestione per nuove richieste. Come nel caso precedente, si prende in esempio una funzionalità già implementata per identificare quali sono i blocchi principali da aggiungere per inglobare la gestione di una nuova richiesta. In questo caso viene preso in analisi il caso di aggiunta di nuove politiche alla configurazione di PF. Il file contenete l'API di gestione del servizio web è **pf_webservice/source/api/v1.py**. Per creare una mappatura tra la funzione da eseguire ed il path su cui essa viene invocata si utilizza l'annotazione `@v1.route(...)`. Un esempio viene fornito da

```
@v1.route("/setRules/v1", methods=['POST'])
def set_rules():
    try:
        payload = request.get_data()
        xmlvalidator.validate_input(payload)
        pfwrapper.add_rules(payload)
        return ok_message()
    except Exception as e:
        print e
        return error_message(e)
```

in questo modo la funzione definita risponde alle richieste inviate ad uno specifico percorso (in questo caso "/setRules/v1"). Viene inoltre specificato il metodo di richiesta atteso per eseguire tale funzione (in questo caso "POST").

Oltre a poter estendere il comportamento del servizio web rendendo più ricca l'API esposta, è possibile estendere il comportamento delle funzionalità esistenti. Il caso di maggior interesse è individuato nella funzione mostrata a titolo di esempio. Essa rappresenta il cuore della vNSF poichè gestisce l'aggiunta delle regole al packet filter. La richiesta ricevuta viene preliminarmente

validata utilizzando un XML Schema. La funzione di validazione è realizzata all'interno dello script **pf_webservice/source/xsd/xmlvalidator.py**. Ciò che risulta però di maggiore interesse ai fini dell'estensione del web service è il contenuto del file **pf_webservice/source/xsd/mspl.xsd**. In questo file sono infatti presenti le definizioni dei campi utilizzabili per specificare la policy in formato MSPL. Poichè la richiesta viene validata utilizzando il file citato, per aggiungere nuovi campi è necessario modificare l'XML Schema in modo che essi siano supportati. Un esempio è fornito dalla seguente dichiarazione

```
...
<simpleType name="ip">
    <annotation>
        <documentation>An IP. It can be a single IP, a range or an IP in CIDR
            form.
        </documentation>
    </annotation>
    <restriction base="string">
        <pattern value="(\d+\.\d+\.\d+\.\d+(-\d+\.\d+\.\d+\.\d+)?)
            |(\d+\.\d+\.\d+\.\d+/\d+)|\*|any" />
    </restriction>
</simpleType>
...
<element maxOccurs="1" minOccurs="0" name="source-address" type="tns:ip">
    <annotation>
        <documentation>The source IP address. It can be a single IP, a range
            or an IP in CIDR form.</documentation>
    </annotation>
</element>
...
```

in tale esempio viene definito il tipo "ip" atteso all'interno della richiesta, per il quale viene specificato il formato atteso. Viene successivamente dichiarato l'elemento "source-address" per il quale viene specificato il tipo ed il numero di occorrenze possibili. Risulta dunque necessario dichiarare i nuovi campi che si desidera utilizzare durante la specifica della policy MSPL all'interno di questo file.

Con la modifica dell'XML Schema risulta possibile utilizzare nuovi elementi all'interno della richiesta. Resta da implementare la parte di gestione del campo aggiunto. La funzione di traduzione delle policy da formato MSPL a regola di PF viene realizzata all'interno del file **pf_webservice/source/pffun/mspl_translator.py**. Per completare l'esempio relativo all'elemento "source-address", è possibile evidenziare all'interno della funzione di traduzione il seguente codice

```
def xml_to_pfrule(data):
    print "starting translation"
    print data
    xml_datas = []
    xml_rules = xmltodict.parse(
        data,
        dict_constructor=dict
        )["mspl-set"]["it-resource"]["configuration"]["rule"]
    if isinstance(xml_rules, list):
        for xml_rule in xml_rules:
            xml_datas.append(xml_rule)
    else:
        xml_datas.append(xml_rules)
    rate_limit = None
    rules = pf.PFRuleset()
    for xml_data in xml_datas:
        rule = pf.PFRule()
        ....
```

```
xml_src = getValIfKeyExists(
    xml_data["condition"]["packet-filter-condition"], "source-address")
if xml_src is not None:
    Addrs = pf.PFAddr(xml_src)
....
if Addrs is not None and Ports is not None:
    rule.src = pf.PFRuleAddr(Addrs, Ports)
elif Addrs is not None:
    rule.src = pf.PFRuleAddr(Addrs)
elif Ports is not None:
    rule.src = pf.PFRuleAddr(pf.PFAddr(), Ports)
....
```

per modellare le regole di PF viene utilizzato `py-pf`, il quale fornisce un modulo Python che incapsula le regole di PF. Per maggiori dettagli su tale modulo si rimanda alla guida raggiungibile al link http://www.kernel-panic.it/programming/py-pf/. Al termine di tali operazioni è possibile utilizzare a pieno le estensioni desiderate sul web service.

Bibliografia

[1] FBI, "2018 Internet Crime Report" https://pdf.ic3.gov/2018_IC3Report.pdf

[2] Cui, Chunfeng and Deng, Hui and Telekom, Deutsche and Michel, Uwe and Damker, Herbert, "Network Functions Virtualisation" http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[3] L. Grossi, E. Maffione, G. Marasso, S. Ruffino, "SDN e NFV: quali sinergie?", Notiziario Tecnico Telecom Italia, Vol. 2, 2014, pp. 44-65

[4] ETSI, "Network Functions Virtualisation (NFV); Use Cases", May 2017, http://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf

[5] Mell, P. and Grance, T. "The NIST definition of cloud computing", National Institute of Standards and Technology, September 2011, http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[6] Hu, F., Hao, Q. and Bao, K. "A survey on software-defined network and openflow: From concept to implementation", IEEE Communications Surveys & Tutorials, Vol. 16, No. 4, May 2014, pp. 2181-2206, DOI 10.1109/COMST.2014.2326417

[7] Lara, A., Kolasani, A. and Ramamurthy, B. "Network innovation using openflow: A survey.", IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, August 2013, pp. 493-512, DOI 10.1109/SURV.2013.081313.00105

[8] Jarschel, M., Zinner, T., Hossfeld, T., Tran-Gia, P. and Keller, W. "Interfaces, attributes, and use cases: A compass for SDN.", IEEE Communications Magazine, Vol. 52, No. 6, June 2014, pp. 210-217, DOI 10.1109/MCOM.2014.6829966

[9] R. Finlayson, "IP Multicast and Firewalls", RFC-2588, May 1999, DOI 10.17487/RFC2588

[10] Scarfone, K. and Hoffman, P. "Guidelines on Firewalls and Firewall Policy", NIST Special Publication, Vol. 800, September 2009

[11] The netfilter.org project, https://netfilter.org/

[12] OpenBSD PF - User's Guide, https://www.openbsd.org/faq/pf/

[13] pfSense® - World's Most Trusted Open Source Firewall, https://www.pfsense.org

[14] IPFire, https://www.ipfire.org/

[15] Kernel Virtual Machine, https://www.linux-kvm.org/page/Main_Page

[16] ab - Apache HTTP server benchmarking tool, https://httpd.apache.org/docs/2.4/programs/ab.html

[17] CentOS, HowTos Network IPTables http://wiki.centos.org/HowTos/Network/IPTables

[18] Hansteen, Peter NM, "The book of PF: a no-nonsense guide to the OpenBSD firewall", 2014, No Starch Press, ISBN: 978-1-59327-589-1

[19] Buechler, Christopher M., Jim Pingle, "pfSense: The Definitive Guide.", 2009 Reed Media Services

[20] Nginx, Wiki main https://www.nginx.com/resources/wiki/

[21] Using th Apache Benchmark Utility https://www.ibm.com/support/knowledgecenter/en/SS8NLW_9.0.0/com.ibm.swg.im.infosphere.dataexpl.engine.best-prac.doc/c_bp-results-benchmark-apache.html

[22] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser, "Terminology for policy-based management", RFC-3198, November 2001, DOI 10.17487/RFC3198

[23] B. Moore, E. Ellesson, J. Strassner and A. Westerinen, "Policy core information model–version 1 specification", RFC-3060, February 2001, DOI 10.17487/RFC3060

[24] F. Valenza and A. Lioy, "User-oriented Network Security Policy Specification", J. Internet Serv. Inf. Secur., Vol. 8, No. 2, May 2018, pp. 33-47, DOI 10.22667/JISIS.2018.05.31.033

[25] SHIELD: Securing against intruders and other threats through a NFV-enabled environment, https://www.shield-h2020.eu
[26] G. Gardikis, S. Pantazis, G. Kolonias, R. Preto, C. Fernandez, B. Gaston, P. Adamidis, A. Litke, N. Papadakis, D. Papadopoulos, G. Dimopoulos, M. De Benedictis, O. Segou and J.N. Mendoza, "Updated specifications, design, and architecture for the usable information driven engine", March 2018, https://www.shield-h2020.eu/documents/project-deliverables/SHIELD_D4.2_Updated_specifications,_Design_and_Architecture_for_the_Usable_Information-Driven_Engine_v1.0.pdf
[27] The SECURED project, https://www.secured-fp7.eu
[28] OSM, https://osm.etsi.org
[29] OSM Information Model, https://osm.etsi.org/wikipub/index.php/OSM_Information_Model
[30] Apache Kafka, https://kafka.apache.org/
[31] cloud-init Documentation, https://cloudinit.readthedocs.io/en/latest/
[32] bsd-cloudinit, https://pellaeon.github.io/bsd-cloudinit/
[33] OpenStack, https://wiki.openstack.org/wiki/Main_Page
[34] JAAS - Juju as a Service, https://jaas.ai
[35] Ansible is Simple IT Automation, https://www.ansible.com
[36] Nmap: the Network Mapper, https://nmap.org/
[37] Flask — The Pallets Projects, https://www.palletsprojects.com/p/flask/
[38] iPerf - The TCP, UDP and SCTP network bandwidth measurement tool, https://iperf.fr/
[39] G. Adelson-Velskii, E.M. Landis, "Odin algoritm organizacii informacii", Doklady Akademii Nauk SSSR, 146:263-266, 1962
[40] D. Hartmeier, A.G. Systor, "Design and Performance of the OpenBSD Stateful Packet Filter (pf).", USENIX Annual Technical Conference, FREENIX Track, June 2002, pp. 171-180
[41] OSM Release FIVE, https://osm.etsi.org/wikipub/index.php/OSM_Release_FIVE