POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Configurazione di vNSF tramite sistemi basati su intenti

Relatori
prof. Antonio Lioy
ing. Ignazio Pedone
ing. Daniele Canavese

Candidato
Lorenzo Mauriello

Anno accademico 2021-2022

Un ringraziamento
speciale alla mia
Famiglia e a tutti coloro
che mi hanno sostenuto.

Indice

Capitolo 1

Introduzione

I cambiamenti tecnologici legati alla digitalizzazione dell'informazione hanno modificato il comportamento sociale di molte persone, creando di fatto un binomio inscindibile tra individuo e tecnologia. La stretta interazione individuo-tecnologia ha luogo grazie all'avvento di Internet, o meglio del World Wide Web[1], nel 1991[2] per il grande pubblico, stravolgendo il modo in cui le persone comunicano e conservano le informazioni. Secondo uno studio dell'Eurostat[3], il costo sempre più abbordabile della tecnologia unito alla capillarità territoriale delle infrastrutture di telecomunicazione, hanno portato già nel 2007 a connettere ad Internet circa il 53% delle famiglie europee, ciò significa che ogni famiglia possedeva almeno un dispositivo capace di accedere alla rete. La percentuale di famiglie in grado di accedere ad Internet ha continuato ad aumentare, superando il 70% nel 2012 e l'80% nel 2014. Nel 2021, la percentuale di famiglie dell'Unione Europea (UE) con accesso ad Internet è salita al 92%, circa 20 punti percentuali in più rispetto al 2011 e 40 punti rispetto al 2007.
Naturalmente, oltre alle persone comuni, sempre più aziende hanno integrato la tecnologia ed Internet nei propri processi lavorativi e di sviluppo, portando nel 2021[4] il 94% delle aziende ad adottare una connessione fissa a banda larga.

Internet ha consentito alle aziende, oltre di crescere e migliorare, di creare nuovi business completamente incentrati sui servizi offerti attraverso di esso, creando le fondamenta per i cosiddetti servizi cloud[5]. Un esempio di azienda capace di cavalcare l'incredibile successo di Internet è stata Google[6], una delle prime ad offrire

---

[1] https://www.w3.org/

[2] https://www.teche.rai.it/2021/08/30-anni-fa-la-nascita-del-world-wide-web/

[3] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Digital_economy_and_society_statistics_-_households_and_individuals

[4] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Digital_economy_and_society_statistics_-_enterprises

[5] Il cloud computing consiste nella fornitura di servizi di computazione, quali software, database e reti, tramite l'accesso ad una connessione Internet, consentendo agli utenti finali di accedere al servizio offerto ovunque si trovino.

[6] https://about.google/?fg=1&utm_source=google-IT&utm_medium=referral&utm_campaign=hp-header

servizi per la ricerca di informazioni attraverso la rete; Google rientra nella categoria dei fornitori di servizi Software (in inglese Software as a Service o SaaS).
Dopo gli anni 2000, la corsa verso i servizi di cloud compuntig è diventata di principale importanza per molte aziende, basti pensare che, secondo l'Eurostat[7], nel 2021 circa il 40% delle aziende utilizza fortemente servizi legati ad internet e di questi il 73% utilizza sofisticati servizi cloud relativi a software di sicurezza, di hosting per database aziendali, di testing e sviluppo software.
Le aziende spingono sempre di più verso l'utilizzo del cloud compuntig, offrendo servizi non solo per altre aziende, ma anche per i semplici privati. Un esempio banale ed efficace sono i servizi di posta elettronica, questo per far capire quanto siano pervasivi i servizi di cloud computing nella vita quotidiana delle persone.

L'incredibile spinta verso l'utilizzo di internet e del cloud computing porta con se problemi relativi alla sicurezza, infatti internet essendo uno "spazio pubblico di comunicazione" soffre di problemi riguardanti l'accesso doloso ad informazioni private o sensibili, di attacchi che cercano di distruggere i servizi offerti dalle aziende e di attacchi che veicolano le informazioni in modo da ottenerne un tornaconto personale.
Per queste ragioni la sicurezza acquisisce un'elevata importanza per le aziende che offrono i propri servizi tramite Internet, permettendo, nell'ultimo decennio, la nascita dei cosiddetti servizi SECaas (Security-as-a-Service), ovvero una nuova tipologia di servizi basati sul cloud computing che permettono di usufruire di un servizio di sicurezza, completamente configurato e funzionante, con tempi di realizzazione pressoché nulli.
I servizi SECaaS fanno parte della categoria di servizi SaaS ed implementano le loro funzioni su infrastrutture remote sfruttando, nella maggior parte dei casi, la tecnologia definita NFV (Network Function Virtualisation) che offre la possibilità di virtualizzare i servizi di rete, come router, switch o strumenti di gestione del traffico, normalmente eseguiti su hardware proprietario. Utilizzando la tecnologia NFV diventa possibile sfruttare a pieno le risorse offerte dai server remoti, riuscendo a realizzare svariate funzioni di rete chiamate virtual network function (VNF), o meglio nel caso dei servizi SECaas si implementano funzioni di sicurezza di rete come Virtual Private Network, Firewall, Deep packet inspection, Web Application Firewall utilizzabili in modo trasparente e forniti da un Provider, definiti virtual Network Security Function (vNSF).

Quindi, essendo la sicurezza uno dei temi cardine per la società moderna, questa tesi propone la realizzazione di una piattaforma di supporto per utenti non esperti, con il fine di semplificare le configurazioni dei servizi di sicurezza offerti come vNSF, sfruttando quindi oltre alla tecnologia delle NFV anche uno degli approcci più innovativi, presenti nel mondo delle telecomunicazioni, ovvero l'Intent-based Networking [1].
L'Intent-based Networking ha come obiettivo quello di eliminare la componente prettamente tecnica necessaria per la configurazione e personalizzazione dei servizi di rete, impiegando un linguaggio di derivazione naturale, definito appunto intento, per specificare gli obiettivi che le configurazioni devono rispettare. Così facendo,

[7] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Cloud_computing_-_statistics_on_the_use_by_enterprises

ogni utente può esprimere il tipo di protezione di sicurezza che vuole ottenere, attraverso il linguaggio umano, permettendo anche una personalizzazione profonda dei servizi.

Questo lavoro aderisce al progetto europeo FISHY[8] che pone come obiettivo quello di realizzare una serie di sistemi completi di protezione da utilizzare come supporto nelle catene di approvvigionamento nell'ambito dell'Information and Communication Technologies[9] (ICT). In particolare, questa tesi si dedica alla realizzazione di un sistema multi-piattaforma di gestione degli intenti in grado di configurare, nel contesto dei servizi vNSF, un Web Application Firewall (WAF), cioè una tecnologia che aumenta la protezione delle applicazioni web aziendali e aiuta le organizzazioni a difendersi adeguatamente da diversi tipi di attacchi informatici, proteggendo in maniera esaustiva i dati grazie alla capacità di analizzare gli accessi agli asset del sistema su cui è configurato.
Il lavoro prevede l'espansione della piattaforma NEMO[10], implementata da OpenDaylight[11], in grado di accettare in ingresso intenti e di tradurli in configurazioni concrete, in modo del tutto trasparente all'utente.

La tesi si suddivide nei seguenti capitoli:

- Capitolo 2: analisi della tecnologia NFV secondo lo standard;
- Capitolo 3: introduzione dell'approccio Intent-Based;
- Capitolo 4: analisi dei principali linguaggi e piattaforme dei sistemi Intent-Based;
- Capitolo 5: concetti fondamentali della tecnologia Web Application Firewall;
- Capitolo 6: progettazione e analisi delle criticità della soluzione;
- Capitolo 7: realizzazione della soluzione;
- Capitolo 8: analisi dei test raccolti dall'implementazione della soluzione;
- Capitolo 9: conclusioni;
- Appendice A: manuale utente;
- Appendice B: manuale sviluppatore.

[8] https://fishy-project.eu/

[9] L'ICT è l'insieme delle tecniche utilizzate nella ricezione, trasmissione ed elaborazione dei dati.

[10] https://test-odl-docs.readthedocs.io/en/latest/user-guide/nemo-user-guide.html

[11] https://www.opendaylight.org/

Capitolo 2

Network Functions Virtualization

In questo Capitolo si presenteranno e analizzeranno i concetti fondamentali per la realizzazione della vNSF, come il paradigma NFV, il Software Defined Networking (SDN) e il Cloud Computing, necessari per fornire una panoramica dettagliata atta a migliorare la comprensione della soluzione proposta ed implementata.

2.1 Panoramica

Il paradigma NFV viene introdotto per la prima volta nel 2012 dall'Europian Telecommunications Institute (ETSI), con lo scopo di rivoluzionare il modo stesso di creare e gestire le reti di telecomunicazione.
Viene scisso il forte legame che intercorre tra hardware e software, presente negli apparati odierni, consentendo lo sviluppo di funzionalità e di servizi di rete come applicazioni software, in modo flessibile ed agile.
Come già detto, grazie a questo paradigma le funzionalità di rete diventano applicazioni software, denominate Virtual Network Function (VNF), che l'operatore può istanziare su server Commodity Off-The-Shelf (COTS), quali ad esempio i classici blade system HP o IBM, sfruttando le tecnologie di virtualizzazione[2]. Ciò viene realizzato tramite l'utilizzo di un livello software di astrazione, denominato strato di virtualizzazione o hypervisor, che permette di creare più macchine virtuali, le cosiddette VM[1], sulla stessa macchina fisica, in grado di eseguire applicazioni differenti nate per diversi sistemi operativi con obbiettivi altrettanto variegati (Figura 2.1).

Le aziende che lavorano nell'ambito delle telecomunicazioni, solitamente, creano delle infrastrutture chiamate Data Center. I Data Center ospitano tutte le apparecchiature che consentono di governare i processi, le comunicazioni e i servizi a supporto di qualsiasi attività aziendale. Queste strutture vengono costruite per semplificare la gestione degli apparati aziendali e ammortizzare i costi dovuti a manutenzione, consumo energetico e sicurezza.
L'organizzazzione degli asset aziendali, così proposta, porta con sè ancora delle limitazioni, sostanzialmente prima dell'adozione di nuovi paradigmi come quello NFV,

[1]Virtual Machine

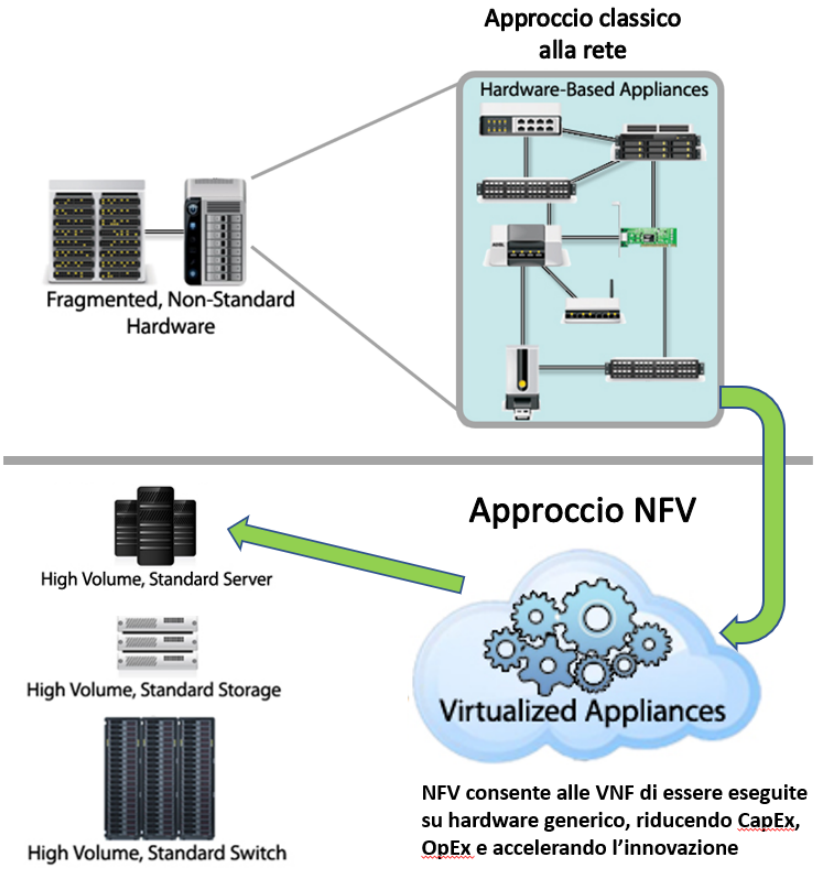


Figura 2.1: Nuovo Paradigma[2]

le configurazioni dei server rispettavano la regola "one application per server"[3], creando un'inefficenza in termini di potenza computazionale impiegata. Questa inefficienza portava ad acquistare molti più server del necessario, generando una gestione poco sostenibile e impattando negativamente dal punto di vista economico.
Per le ragioni sopra elencate il paradigma NFV è stato celermente adottato da tutte le aziende fornitrici di servizi IT[3], portando una vera e propria rivoluzione. Infatti, il meccanismo delle NFV è analogo a quanto avviene oggi per i servizi IT in esecuzione su piattaforme di Cloud Computing, con la differenza che le VNF possono richiedere opportune ottimizzazioni sull'hardware per soddisfare i requisiti di basso ritardo, scalabilità, ridondanza geografica e gestibilità tipici delle reti di telecomunicazioni.
Ricapitolando le aziende ottengono innumerevoli benefici:

- utilizzare la virtualizzazione permette un uso delle risorse più preciso ed accurato;

[2]La reference a questa immagine si può trovare al seguente url https://www.blueplanet.com/resources/What-is-NFV-prx.html

[3]Information Technology

- ottimizzare l'uso delle risorse, quali CPU, memoria, storage e network, attivando sullo stesso server fisico più VM che implementano diverse tipologie di servizio, in questo modo possiamo sfruttare appieno le capacità computazionali delle risorse utilizzando, a parita di potenza, meno server riducendo spese di gestione e, soprattutto, il consumo energetico[4] (consolidamento hardware);

- ampliare o ridurre in modo dinamico la capacità di risorse allocate in base al carico effettivo, migliorando anche le performance complessive (elasticità);

- garantire alta affidabilità, in quanto a fronte di un malfunzionamento hardware, grazie alla virtualizzazione, le VM possono essere facilmente migrate da un server all'altro (resilienza);

- gestire più tenant contemporaneamente, consentendo agli operatori di fornire al singolo utente servizi specifici sullo stesso hardware, ottenendo comunque un livello di sicurezza e isolamento alti, sempre garantiti dalla virtualizzazione;

- ridurre il Total Cost of Ownership (TCO) e migliorare il Time-to-Market, sfruttando la maggiore agilità e flessibilità offerta da NFV nel dispiegamento dei servizi.

2.1.1 NFV, SDN e Cloud Computing a confronto

Il concetto di NFV abbraccia un ampio spettro di tecnologie e sistemi, ragion per cui, inevitabilmente, va a legarsi ad altri settori ed in particolare a quelli del Cloud Computing e del Software Defined Network (Figura 2.2). Cercheremo quindi di confrontare le diverse tecnologie per rendere chiare e comprensibili le differenze.

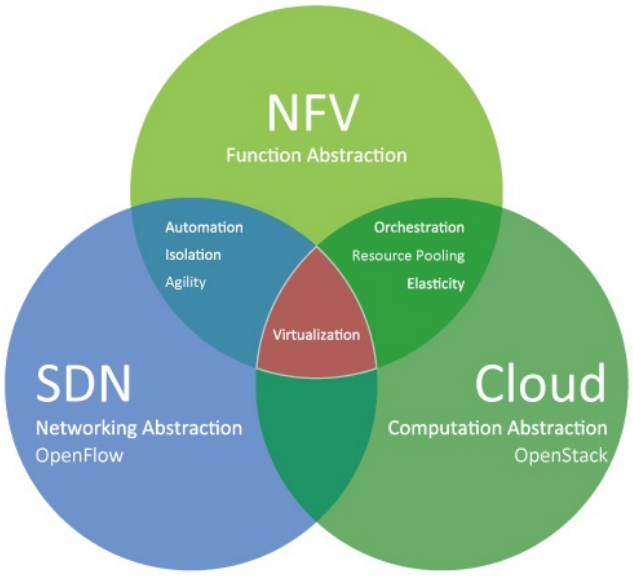


Figura 2.2: Tecnologie a Confronto [4]

Il National Institute of Standards and Technology (NIST) definisce il Cloud Computing come un modello per abilitare un accesso, disponibile ovunque, conveniente e on-demand[5], ad un pool di risorse configurabili che possono essere facilmente assegnate e rilasciate con il minimo sforzo di gestione da parte del provider.[5]
Il Cloud Computing è, di fatto, un generico conglomerato di servizi centralizzati, fruibili remotamente da un utente, basati sulla virtualizzazione; per comprendere meglio il concetto di Cloud Computing, però, è necessario specificare i modelli di servizio su cui si basa:

- *Software as a Service (SaaS)*: all'utente viene fornita la possibilità di utilizzare applicazioni di un provider terzo, eseguite in remoto accessibili tramite web browser;
- *Platform as a Service (PaaS)*: all'utente viene fornita la possibilità di istanziare applicazioni create da lui stesso, sull'infrastruttura cloud remota.
  Gli applicativi per essere istanziati 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)*: all'utente viene fornita la possibilità di utilizzare risorse computazionali, di storage o di altro tipo per poter eseguire software arbitrario.

Compreso cosa si intende per Cloud Computing è quasi spontaneo accomunare i concetti esposti con le NFV, infatti anche queste ultime possono essere viste come un'esposizione di un conglomerato centralizzato di servizi. La differenza tra questi due concetti sta nel tipo di servizi su cui si concentrano e su come si utilizza la virtualizzazione per utilizzarli.
Per quanto riguarda la tipologia di servizi, le NFV si limitano a concentrarsi sulle funzioni di rete, invece il cloud compuntig offre servizi general purpose virtualizzando risorse computazionali. Ad esempio, le VNF costituiscono solo un sottoinsieme del modello di servizio SaaS, ciò nonostante Il cloud ricopre un ruolo fondamentale nello sviluppo di funzioni virtualizzate, basti pensare al modello IaaS.
Grazie al modello IaaS è possibile accedere a risorse virtualizzate costituendo un'ambiente di sviluppo facilmente scalabile, fessibile e poco costoso che risulta ideale per lo sviluppo di nuove funzioni virtualizzate.
L'unione dei due paradigmi ci permette di implementare e sperimentare velocemente nuove VNF effcienti, contribuendo a spostare il mondo delle telecomunicazioni verso un ambiente sempre più virtualizzato.

L'SDN, proposto per la prima volta da Nicira Networks, è un paradigma che fornisce una gestione scalabile, resiliente, dinamica e sicura della rete, creando un disaccoppiamento tra le funzioni di control plane e data plane[6].

[4]La reference a questa immagine si può trovare al seguente url https://www.cas-well.com/applications/sdn-nfv-cloud-computing.html

[5]Un utente può unilateralmente usufruire di capacità computazionali al bisogno, senza richiedere intervento umano.

Con data plane si intende il componente di sistema che ha il compito di ricevere i pacchetti di cui è composto il traffico IP, processarli e inoltrarli nel modo più veloce possibile; invece, con control plane si intende il componente di sistema che ha il compito di definire il comportamento del data plane, stabilendo le regole che il singolo dispositivo utilizza per inoltrare i pacchetti.
Con questo paradigma l'amministratore di rete di un Data Center può ottenere un maggior controllo sulla rete incrementando le performace e l'efficenza delle proprie risorse.
A differenza del Cloud Computing, qui, la relazione tra NFV e SDN si basa molto sul loro grado di interoperabilità, piuttosto che sulla loro somiglianza. Di fatto, mentre la prima si occupa di fornire servizi virtualizzati, la seconda si occupa della creazione di percorsi di rete, guidata da un controllore centralizzato. Il sillogismo che si sviluppa dalle precedenti affermazioni ci porta ad afferire che l'NFV è in grado di sfruttare le capacità di gestione del traffico del paradigma SDN, prorprio perchè quest'ultimo è in grado di eseguire un forwarding del traffico stesso attraverso differenti funzioni.
Entrambe le tecnologie forniscono un approccio inedito e innovativo, tutta questa innovazione si ripercuote nel modo di sviluppare le applicazioni. Per quanto riguarda l'SDN le applicazioni devono essere riscritte per poter sfruttare la separazione tra control e data plane, invece nel caso delle NFV devono essere implementate exnovo rispettando i nuovi criteri di virtualizzazione. Ancora, entrambe, seppur in modo e misura differente, traggono vantaggio dalla capacità dell'hardware utilizzato, ad esempio l'SDN trae vantaggio dalla grossa capacità d'input e output, invece l'NFV dalla grande potenza di calcolo.

Come abbiamo letto i paradigmi di SDN e Cloud Computing si intrecciano al paradigma di NFV, portando in auge un concetto di base che li accomuna e differenzia, ovvero la virtualizzazione.
La virtualizzazione dell'SDN è primariamente diretta alla rappresentazione di risorse, mentre quella NFV è diretta a separare le funzionalità dall' infrastruttura e per finire il Cloud Computing sfrutta la virtualizzazione per fornire un pool di servizi eterogenei.
Il concetto di virtualizzazione risulta quindi fondamentale e verrà analizzato più volte nel testo.

2.2 Architettura NFV standard ETSI

In questa sezione approfondiremo l'analisi del paradigma NFV, in particolare andremo ad analizzare dettagliatamente l'organizzazione della sua architettura secondo i dogmi dettati dello standard ETSI[7].

Come primo passo possiamo osservare l'organizzazione dell'architettura ad alto livello, mostrata anche in Figura 2.3, che può essere riassunta in tre moduli fondamentali:

- *Il modulo Virtualized Network Functions (VNFs)* è l'implementazione software di una Network Function (NF[6]) che solitamente è realizzata per essere eseguita in un'infrastruttura virtualizzata;

- *Il modulo NFV Infrastructure (NFVI)* rappresenta l'insieme delle risorse messe a disposizione delle VNFs. L'NFVI, quindi, si occupa di gestione e fornire le risorse computazionali, di rete e storage dati tramite uno strato di virtualizzazione;

- *Il modulo NFV Management and Orchestration (NFV-MANO)* permette l'implementazione e la gestione delle VNFs regolandone il ciclo di vita.

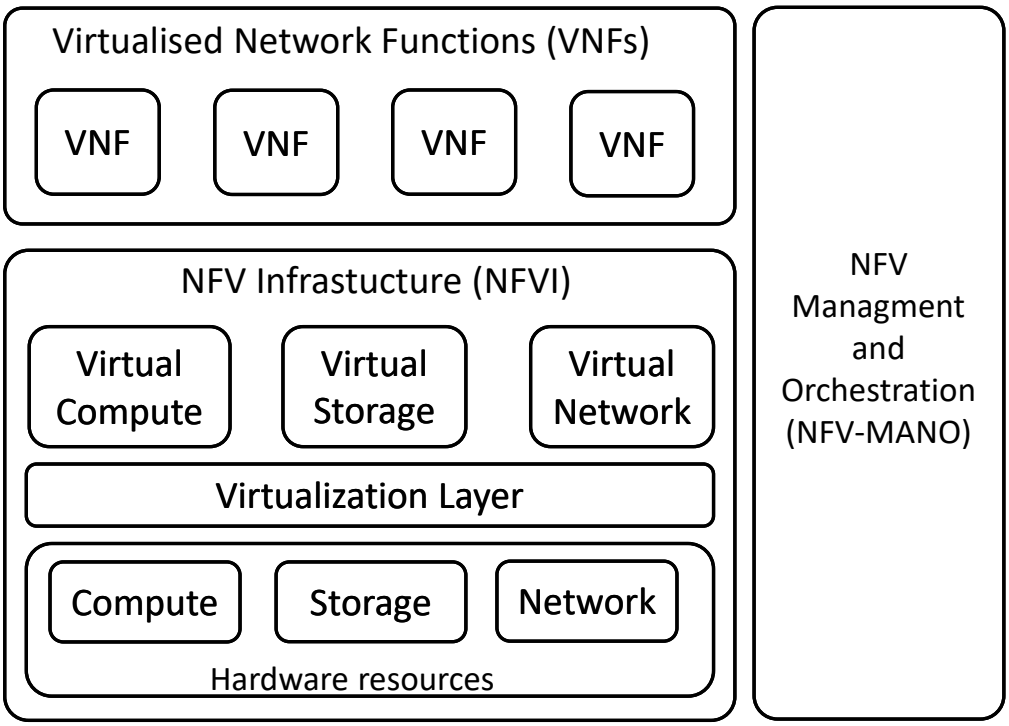


Figura 2.3: Vista ad alto livello Framework NFV[7]

Questi moduli a loro volta possono essere scomposti e dettagliati, come possiamo osservare nella seconda Figura proposta 2.4, infatti cercheremo di analizzare il tutto in modo sintetico ed esaustivo, seguendo l'ordine archutetturale definito dalla figura.

Procedendo dall'alto verso il basso la prima componente che incontriamo è l'OSS/BSS che ha il compito di fornire supporto all'intero sistema; in particolare l'Operations Support System (OSS) interviene nella configurazione di rete, nella gestione

[6]la Network Function definisce un blocco funzionale dove le interfacce sono ben definite e possiedono un comportamento noto a priori.

[7]La reference a questa immagine si può trovare nello standard [7]

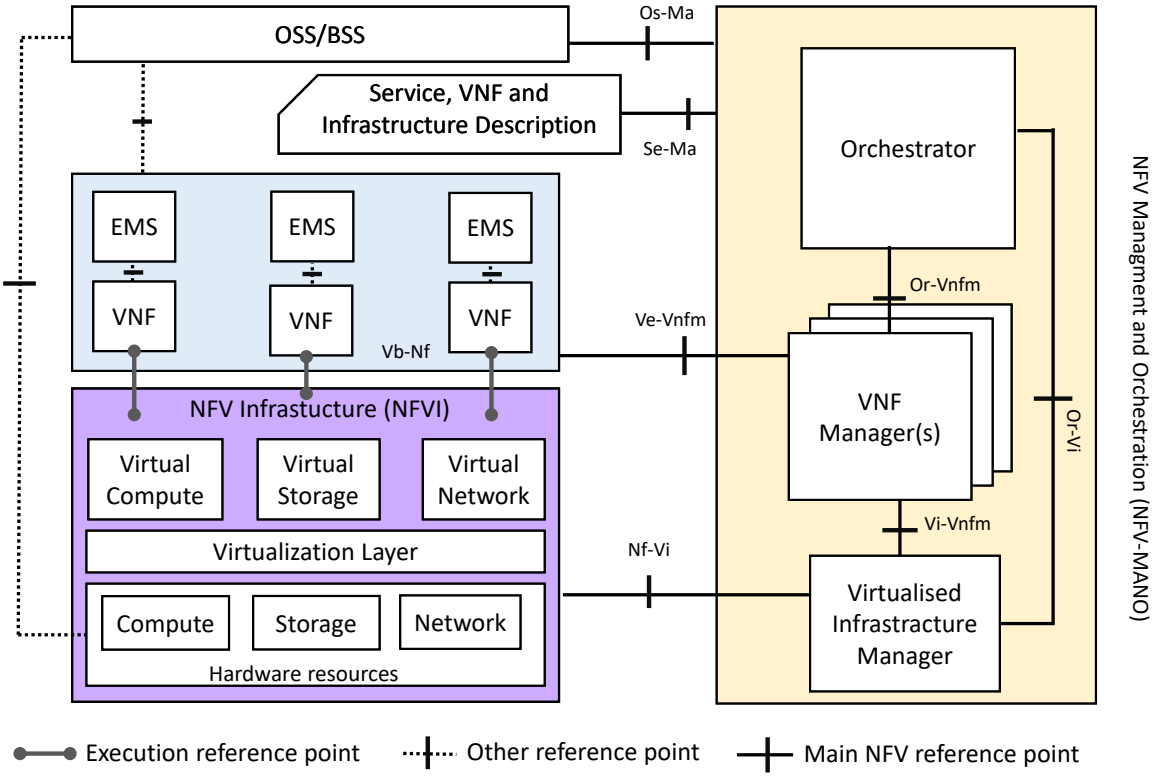


Figura 2.4: Vista ad alto livello Framework NFV[8]

dei casi di malfunzionamento e nella fornitura del servizio. Invece, il Business Support System (BSS) è orientato al supporto commerciale dell'azienda, in particolare si occupa di verificare che gli accordi cliente-fornitore siano rispettati. Di solito queste componenti sono divise ma perfettamente integrate tra loro per monitorare e gestire i servizi offerti, dipendono fortemente dalle decisioni del fornitore che implementa l'architettura, ragion per cui, in questo caso, lo consideriamo un elemento "marginale" seppur importantissimo.
La componente Element Management System (EMS) è direttamente collegata all'OSS/BSS e si occupa della gestire di una VNF. Il collegamento è fondamentale, di fatto l'EMS opera secondo le direttive dell'OSS/BSS, questo perchè, come già accennato, tra cliente e fornitore di servizio esistono dei contratti che devono essere rispettati, quindi un'attività che il modulo può svolgere è quella di mantenere un determinato grado di Quality of Service (QoS).

La componente Service, VNF and Infrastructure Description definisce il così detto data model, ovvero le caratteristiche del servizio di rete riguardante la VNF. Grazie al modello offerto al modulo MANO si rende possibile eseguire il deploy della funzione, in particolare nel data model vengono conservati VNF deplyment template, VNF Forwarding Graph, servizi correlati alla VNF e per finire il modello informativo della infrastruttura.

[8]La reference a questa immagine si può trovare nello standard [7]

Una componente chiave dello schema proposto è il Virtualization Layer incaricato di astrarre e partizionare logicamente le risorse, nonchè di realizzare la separazione tra il software e l'hardware fisico sottostante. Naturalmente la componente coincide con il concetto di Hypervisor, ma si ci astiene dall'indicarne uno in particolare, lasciando molta libertà di implementazione.
In generale questa è una componente critica, infatti deve fornire supporto a diverse tecnologie di virtualizzazione per garantire un rapporto efficienza-prestazione molto elevato, deve generare e aggregare diverse risorse virtuali, attraverso interfaccie standard e ben definite, per garantire il supporto ad ogni tipo di NF che può trovarsi all'interno della rete di un operatore.

Le restanti componenti non possono essere sintetizzate dal momento che appartengono ai moduli principali dell'architettura, per queste ragioni li approfondiremo nelle prossime sezioni.

2.3 Architettura VNF

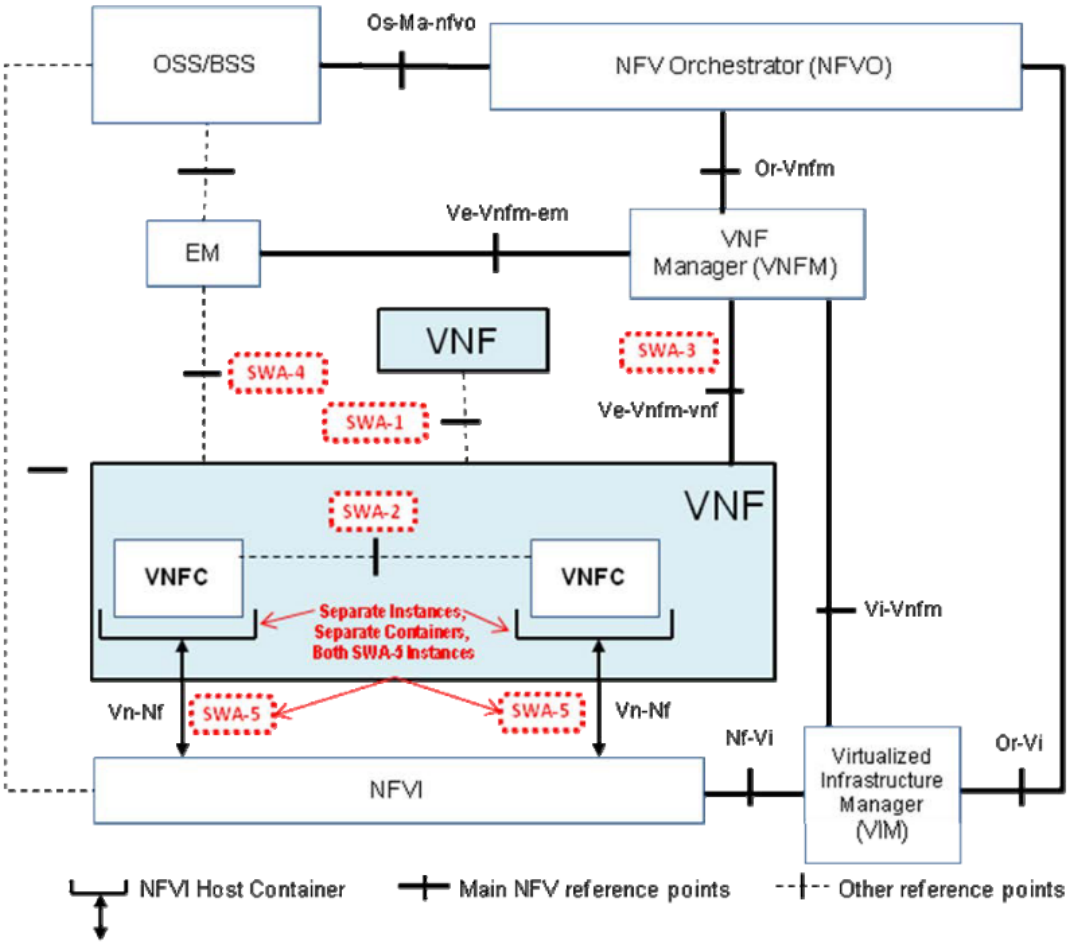


Figura 2.5: VNF interfacce e architettura del Framework NFV (fonte: [8])

In questa sezione abbassiamo il livello di astrazione dell'architettura e analiziamo nel dettaglio il modulo Virtual Network Functions riferendoci allo standard ETSI [8] che gli è stato dedicato.

Una VNF fornisce l'implementazione software di una funzione di rete, la quale deve essere eseguita sulla NFVI e gestita durante il suo ciclo di vita da un orchestratore ed un gestore NFV-MANO, ossia dal NFV Orchestrator (NFVO) e dal NFV Manager (NFVM). Le VNF, allo scopo di interagire con NFVO e NFVM, seguono un'architettura rigida definita da interfacce che permettono sia l'interazione interna alla VNF, che l'interazione con altri componenti del paradigma NFV; tale approfondimento è mostrato nella Figura 2.5.

Come già detto, l'architettura della VNF deve rispettare requisiti abbastanza rigidi in termini di performance, scalabilità, resilienza, sicurezza e funzionalità, per questo è stata progettata per essere organizzata in componenti modulari, che prendono il nome di Virtual Network Function Component (VNFC Figura 2.6).
In questo caso una VNFC è un'entità software sviluppata all'interno di un continer fornito dal virtualization layer, ciò vuol dire che una singola VNF può essere creata seguendo una struttura monolitica, cioè integrando la logica di Control plane, data plane e managment in un unico blocco, oppure si può comporre da più VNFC, le quali sono contenute in singoli container in grado di eseguire una comunicazione multinterfaccia. I VNFC che compongono la singola funzione virtualizzata possono

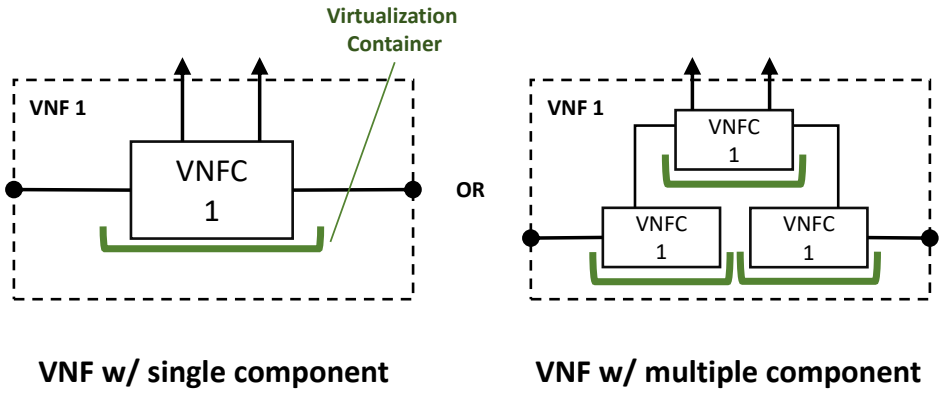


Figura 2.6: Organizzazione dei VNF Components[9]

anche essere parallelizzabili, dando quindi la possibilità di essere istanziati più volte per una singola VNF.
Anche in questo caso, una singola VNF non è legata ad una tecnologia specifica, ma può essere implementata in modo differente da diversi fornitori. La comunicazione multinterfaccia mostrata nella Figura 2.5 chiarisce in modo esaustivo l'utilizzo che ETSI propone di queste interfacce. Ogni "collegamento" tra moduli funzionali è caratterizzato dalla presenza di Reference point; in totale possiamo contarne cinque:

- *SWA-1*: si riferisce all'insieme delle interfacce che abilitano la comunicazione tra le VNF, sia all'interno dello stesso servizio di rete, sia tra servizi differenti.

[9]L'immagine utilizza come reference le immagini e le informazioni presenti nello standard [8]

L'interoperabilità tra le VNF ci consente di ottenere funzionalità di rete più complesse;

- *SWA-2*: si riferisce all'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 SWA-2 è rappresentato dalle performance, in particolare si vogliono minimizzare le latenze. Esse sono solitamente trasparenti agli utilizzatori della VNF, seppur non standard e decise dal provider;
- *SWA-3*: corrispondente al reference point Ve-Vnfm-vnf, l'interfaccia SWA-3 permette la comunicazione tra la VNF e il modulo di gestione, contenuto nel componente NFV-MANO;
- *SWA-4*: utilizzata dal modulo Element Management (EM) per comunicare con una VNF e per gestire la stessa durante l'esecuzione in armonia con il modello di business del provider;
- *SWA-5*: corrisponde al riferimento Vn-Nf e costituisce una rappresentazione logica di più interfacce. In generale, il compito di questo reference point è quello di fornire comunicazione tra la VNF e l'infrastruttura NFVI, operando sulle differenti tipologie di risorse che il virtualization layer fornisce.

2.3.1 vNSF

Per concludere con l'argomento delle VNFs, introduciamo una specializzazione di queste ultime, ovvero le Virtual Networik Security Functions (vNSF).
Le vNSF non sono nient'altro che una specializzazione delle VNF in funzioni relative alla sicurezza della rete (es. firewall, IDS, etc..), le quali si distinguono in passive o attive.
Le vNSF passive vengono utilizzate per monitorare la rete e raccogliere dati sull'infrastruttura, le attive vengono utilizzate in risposta ad una minaccia informatica, in modo da contrastare il pericolo. Il concetto di vNSF si inserisce, in un contesto intermedio, tra la gestione e la prevenzione dei rischi in una rete generica e può essere implementata utilizzando la tecnologia di virtualizzazione di funzioni di rete NFV, descritta nelle sezioni precedenti.
Questo approccio alle funzioni di sicurezza di rete rivoluziona il mercato e il design stesso della rete; infatti, prima dell'introduzione di questo concetto, la struttura era fornita di sottoreti ad hoc per ospitare questo tipo di servizi. Il passo definitivo verso questo approccio è stato determinato dalla diffusione della tecnologia NFV che elimina, di fatto, ogni apparato fisico e molti vincoli sul design della rete, favorendo la nascita del paradigma Security as a Service (SECaaS [9]).

2.4 Architettura NFVI

Il prossimo step all'interno dell'architettura NFV, delineata da ETSI [10], riguarda l'infrastruttura denominata NFVI introdotta all'inizio della Sezione 2.2.

Come in precedenza, abbassiamo il livello di astrazione per concentrarsi sulla composizione del modulo NFVI, rappresentato in modo esaustivo in Figura 2.7. Pos-

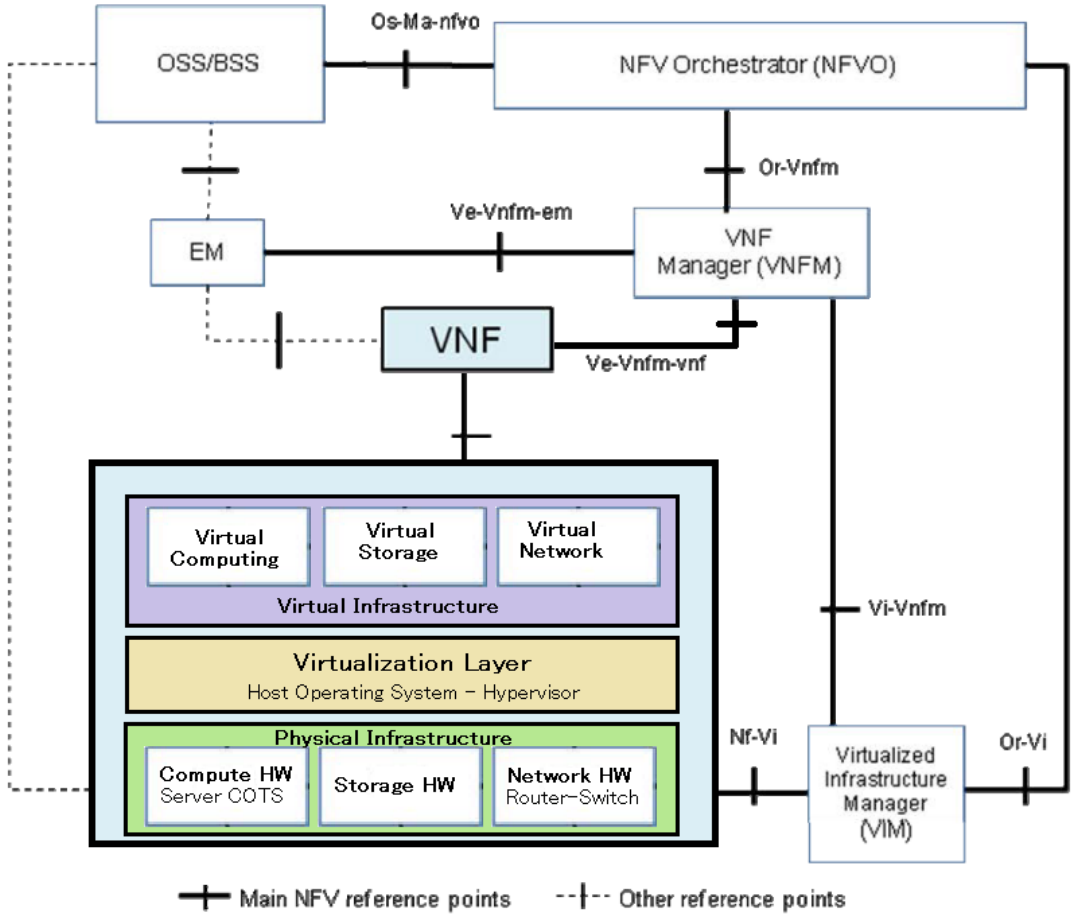


Figura 2.7: Organizzazione modulo NFVI[10]

siamo notare la suddivisione del modulo in tre componenti fondamentali, che a loro volta si compongono da diversi elementi in grado di interaggire tra loro.

La prima componente che incontriamo è la Physical Infrastructure che rappresenta le risorse concrete di tutta l'impalcatura virtualizzata. A questo livello si possono evidenziare tre tipologie di elementi distinti, che coincidono con le risorse hardware; nello specifico troviamo risorse computazionali, risorse di memorizzazione e risorse di rete.
Le risorse computazionali sono hardware COTS non ottimizzato per un particolare uso, ma general purpose; in questo contesto si fa riferimento a server standard che utilizzano hardware commerciale. I dispositivi che forniscono le risorse di memoria solitamente si adattano al paradigma Storage Area Network (SAN) oppure al paradigma Network Attached Storage (NAS). Infine, le risorse di rete includono tutti i dispositivi basilari di cui una rete necessita come switch, router, ma anche link di tipo diretto o wireless.

[10]L'immagine utilizza come reference le immagini e le informazioni presenti nello standard [10]

La seconda componente è il Virtualization Layer, questa coincide con lo strato intermedio del modulo NFVI ed è costituito da un hypervisor.
L'hypervisor, noto anche come monitor della macchina virtuale, è un processo che crea e gestisce le macchine virtuali (VM),cioè crea disaccoppiamento tra hardware e software permettendo ad un infrastruttura fisica di supportare più VM guest attraverso la condivisione virtuale delle risorse.

L'ultima componente che incontriamo è la Virtual Infrastructure che contiene la rappresentazione virtuale delle risorse che l'hypervisor mette a disposizione alle VNFs.
Qui troviamo un parallelismo diretto con la Physical Infrastructure, infatti si distinguono tre entità, ovvero il Virtual Computing che rappresenta le VM, il Virtual Storage che coincide con file, volumi o altri oggetti di storage e per finire la Virtual Network che coincide con router, switch o link virtuali.

Dalle indicazioni dello standard, oltre alla compartimentazione precedente, si delineano altri tre domini di lavoro distinti, quali:

- *Compute Domain*: fa parte di questo dominio l'hardware COTS, in particolare si ci riferisce alle CPU, alle scheda di rete e allo storage che compongono un server. Si può identificare questo dominio con parte del modulo *Physical Infrastructure*, escludendo però le risorse di rete;

- *Hypervisor Domain*: fa parte di questo dominio il *Virtualization Layer* e le componenti di computing e storage del modulo *Virtual Infrastructure*.
  L'obiettivo che si prefigge è di effettuare una mediazione tra le risorse del Compute Domain e le macchine virtuali delle funzioni software.
  Grazie alla virtualizzazione delle risorse si può emulare qualsiasi tipo di hardware sottostante, al punto che una macchina virtuale non sia in grado di distinguere "l'hardware virtuale" da quello fisico;

- *Infrastructure Network Domain*: fa parte di questo dominio sia la componente networking appartenente alla *Physical Infrastructure* che quella appartenente alla *Virtual Infrastructure*. L'obiettivo è quello di fornire una strumentazione completa dal punto di vista della connettività della rete.

2.4.1 NFV tecnologie a confronto VMs vs Containers

Il Virtualization Layer assume un ruolo molto importante nell'architettura NFVI, infatti questa componente impatta in modo diretto sulle performance dell'intero sistema, ragion per cui vale la pena analizzare le teconolige che ad oggi possiamo utilizzare per la sua implementazione.
Come già detto, un'implementazione del Virtualization Layer prevede la presenza di una componente installata al di sopra di un sistema operativo o come parte di esso, chiamata appunto Hypervisor, la quale consente di astrarre le risorse del sistema al fine di eseguire uno o più sistemi operativi virtuali (guests). Il sistema guest definisce una Virtual Machine sulla quale sarà possibile eseguire una VNF.
La principale alternativa a questo approccio è costituita dalla tecnologia di Containerizzazione che fa ricorso ad un'accezione del concetto di virtualizzazione. Qui,

ogni applicazione, insieme al software di supporto necessario, viene incapsulata in un container cercando di garantire isolamento e alte performance. La caratteristica di leggerezza e portabilità dei container deriva dalla capacità di condividere il kernel del sistema operativo dell'host, evitando la necessità di disporre di un sistema operativo distinto per ogni container; cade il concetto di hypervisor ed entra in gioco il concetto di Container Engine che si occupa di gestire ed orchestrare i diversi container instanziati. Grazie a questa soluzione ogni VNF può essere eseguita su un container diverso, senza drenare risore per eseguire un hypervisor, e soprattutto senza ricorrere all'esecuzione di un sistema operativo completo per avviare una singola funzione.

Da quanto emerge nella discussione riguradante le tecnologie implementative, entrambi i sistemi possiedono sia pregi che difetti, ma i container sembrano nettamente più performanti rispetto alle macchine virtuali. Per chiarirci meglio le idee effettuiamo un confronto delle due tecnologie riassumendo il tutto nella Tabella 2.1. Agli albori dello standard ETSI riguardante l'architettura NFV si consigliava fortemente l'adozione di un Hypervisor per l'implementazione delle VNF; con il tempo e vari studi proposti da fonti autorevoli come l'IETF [12], IBM o RedHat si è capito che i container erano la strada più efficiente ed efficace per implementare tale architettura. Per queste ragioni lo standard fu rettificato e nella sua versione definitiva si consiglia l'implementazione delle VNF tramite container.

L'utilizzo dei Container consigliato da ETSI viene adottato anche all'interno della soluzione proposta da questo lavoro, nello specifico verrà utilizzato Docker.

2.4.2 Docker

Docker è una piattaforma open source per lo sviluppo, la distribuzione e l'esecuzione di applicazioni [13]. L'architettura di questo sitema client-server è basata su tre elementi principali:

- *Docker Daemon*: ascolta le richieste dell'API Docker e gestisce oggetti Docker come immagini, containers, reti e volumi. Un deamon può anche comunicare con altri deamon Docker per gestire diversi servizi;
- *Docker Engine REST API*: è un'API utilizzata dalle applicazioni per interagire con il Docker deamon; è accessibile da un client HTTP;
- *Docker CLI*: rappresenta il modo principale in cui molti utenti possono interagire con Docker, ad esempio quando si utilizzano comandi come docker run, il client li invia al Docker Deamon, che li esegue. Bisogna notare che un client Docker può comunicare con più di un deamon.

È stata selezionata questa tecnologia rispetto ad altri competitor sul mercato perchè, tra le tecnologie open source, fornisce le migliori performance, il miglior supporto e la miglior affidabilità anche in termini di sicurezza.

[11] La tabella utilizza le informazioni trovate nel documento [11]

Parametri	Virtual Machines	Containers
Guest OS	Ogni VM gira su hardware virtuale con un kernel caricato in una singola regione di questo hardware	Tutti i containers condividono lo stesso sitema e kernel. Il kernel relativo al container è direttamente inserito in memoria senza traduzioni
Communication	Avviene attraverso dispositivi Ethernet	Avviene tramite meccanismo standard ICP come pipes e sockets
Security	Dipende dall'implementazione dell'Hypervisor	Più problematico, l'isolamento tra container è garantito, ma l'host in qualunque momento può accedere alle operazioni dei container. È necessaria sicurezza aggiuntiva
Performance	Le VM non possono sfruttare il massimo dell'hardware a disposizione, in quanto l'invio di istruzioni macchina dal Guest all'Host soffre di overhead di traduzione	Meno problematico, le prestazioni offerte dai container si avvicinano molto alle performance native offerte dall'Host
Isolation	Difficile o impossibile condividere librerie, file, ecc. tra VM ed Host	Le sottodirectory possono essere montate in modo trasparente e possono essere condivise con facilità
Startup time	le VM sono operative in pochi minuti	I containers sono operativi in pochi secondi
Soragee	le VM hanno bisogno di molto spazio per essere installate e mantenute	I containers prendono una quantità di spazio esigua che può essere condivisa con il sistema ospitante

Tabella 2.1: Comparazione Feature Container vs VM[11]

2.5 Orchestratori e gestori NFV-MANO

In questa sezione si vuole concludere la panoramica sull'architettura NFV parlando dell'ultimo modulo che incontriamo, ovvero del Management and Orchestration module (MANO); nella Figura 2.8, proposta per analizzare l'architettura del modulo, possiamo distinguere tre componenti principali:

- *NFV Orchestrator (NFVO)*;
- *VNF Manager (VNFM)*;
- *Virtualized Infrastructure Manager (VIM)*.

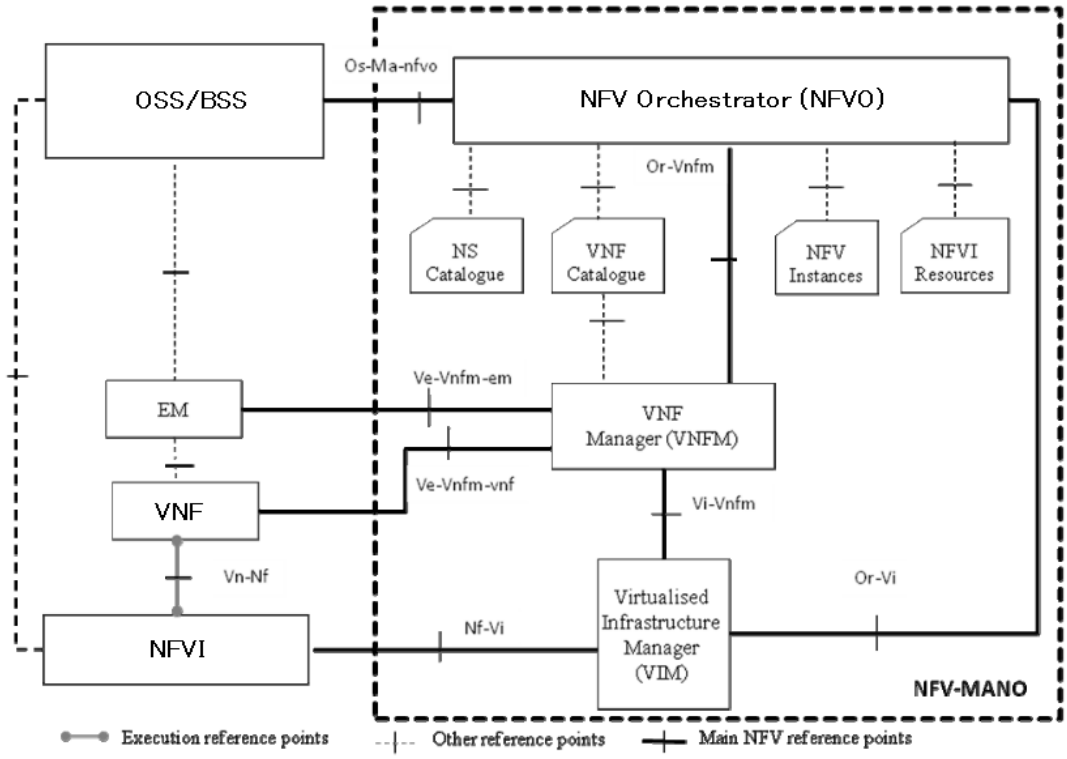


Figura 2.8: Organizzazione modulo NFV-MANO[12]

Il primo blocco che incontriamo coincide con la componente di *Orchestrazione NFV* (NFVO), questo è posto logicamente nel livello più alto dell'architettura ed è responsabile di due operazioni:

- l'orchestrazione delle risorse NFVI su più VIMs;
- la gestione del ciclo di vita completo dei servizi di rete.

Le resposabilità assegnate a questa componente sono potenzialmente molto complesse, ragion per cui lo standard [14], per comodità, le raggruppa in un unico blocco funzionale facedogli condividere una base informativa comune rappresentata dall'NFV Instances e dall'NFV Resources repository. Naturalmente per supportare diverse implementazioni multi-vendor e/o diverse mappature di funzionalità, basate su diversi domini amministrativi, le due features potrebbero essere implementate separatamente.
Le operazioni sopra descritte sono realizzate, rispettivamente, dalle Resource Orchestration functions (RO) e dalle Network Service Orchestration functions (NSO). L'RO offre all'NFVO interfacce di accesso alle risorse virtualizzate, fornite dall'infrastruttura, attraverso un'astrazione che lo rende indipendente dal singolo gestore. Queste interfacce si occupano di gestire le autorizzazioni per richiedere le risorse, sia da parte dei singoli nodi che da parte dei Network Services.

[12]L'immagine utilizza come reference le immagini e le informazioni presenti nello standard [14]

Per quanto riguarda l'NSO ha il compito di coordinare tutte le funzioni fondamentali che compongono un servizio di rete, permettendo di configurarlo ed eseguirlo; ancora, è responsabile della definizione delle policy e della costruzione e gestione della topologia.

Il secondo blocco è rappresentato dal *VNF Manager* responsabile di gestire il ciclo di vita delle VNF istances, viene implementato in più repliche ciascuna dedicata ad una VNF in esecuzione. Questa regola non è rigida, infatti, è possibile raggruppare più VNF dello stesso tipo sotto la gestione di un'unico VNFM.
Si presume che la maggior parte dei task che il VNFM svolge siano funzioni comuni, generiche, applicabili a qualsiasi tipo di VNF; questo si riflette sull'implementazione stessa del blocco, che risulta in gran parte simile in tutte le sue declinazioni. Tuttavia, il framework architetturale NFV-MANO deve poter supportare anche i casi in cui le istanze delle VNF necessitino di funzionalità specifiche per la gestione del loro ciclo di vita, a tale scopo una funzionalità può essere specificata nel VNF Package del VNFM di pertinenza.
L'implementazione e la definizione del funzionamento di una determinata VNF vengono catturati attraverso la definizione di un template, chiamato Virtualized Network Function Descriptor (VNFD), salvato nel VNF catalogue. Il VNFD viene utilizzato dal modulo NFV-MANO per eseguire l'istanziazione di una nuova VNF, seguendo i dettagli ivi descritti; a questo punto, le risorse NFVI sono assegnate ad una VNF in base ai requisiti specificati, ma prendono anche in considerazione requisiti, vincoli e politiche che sono stati pre-forniti o accompagnano la richiesta di istanza (ad es. Politiche dell'operatore, posizionamento della geo-locazione, regole di affinità/anti-affinità, regolamenti locali). È bene ricordare che il VNFM ha accesso ad un variegato repository di pacchetti VNF contenente diverse versioni, della stessa VNF, da eseguire in diversi ambienti di esecuzione o versioni di rilascio dello stesso software. Questo meccanismo permette di istanziare una VNF su qualsiasi infrastruttura, indipendentemente dal tipo di vendor.

L'ultimo blocco che incontriamo è il Virtualized Infrastructure Manager (VIM), esso si occupa di controllare e gestire tutte le risorse che fanno parte del modulo NFVI. Questa componente, di solito implementata all'interno del dominio infrastrutturale di un operatore, può essere dedicata ad un solo tipo di risorsa oppure può essere in grado di gestire risorse computazionali, di storage e di rete contemporaneamente.
Il VIM collabora con gli altri blocchi mediante l'esposizione di due interfacce, una northbound ed una southbound interface capaci di gestire il traffico che viaggia in direzioni ortogonali. La prima fornisce ai livelli superiori le API necessarie per interagire con le risorse hardware, invece, la seconda è legata direttamente all'infrastruttura fisica e permette di riflettere le azioni specificate direttamente sulle risorse.

2.5.1 kubernetes

Kubernetes, annunciato per la prima volata da Google nel 2014, è un sistema open-source di orchestrazione e gestione di container [15] che può essere considerato un esempio di implementazione reale dell'architettura NFV-MANO. Quindi, in questa

sezione andremo ad analizzare l'architettura generale di kubernetes, senza scendere troppo nei dettagli implementativi, in modo da evidenziare i parallelismi con l'architettura NFV-MANO. Per tutti gli approfondimenti e dettagli implementativi non presi in esame nel documento, si consiglia di consultare la documentazione ufficiale [16].

In Figura 2.9 viene proposta una vista ad alto livello dell'architettura di kubernetes dove si distinguono molti elementi, ma noi analizzeremo solo i più importanti. Il sistema è composto dal modulo Control-Plane o Nodo Master, ovvero il Nodo

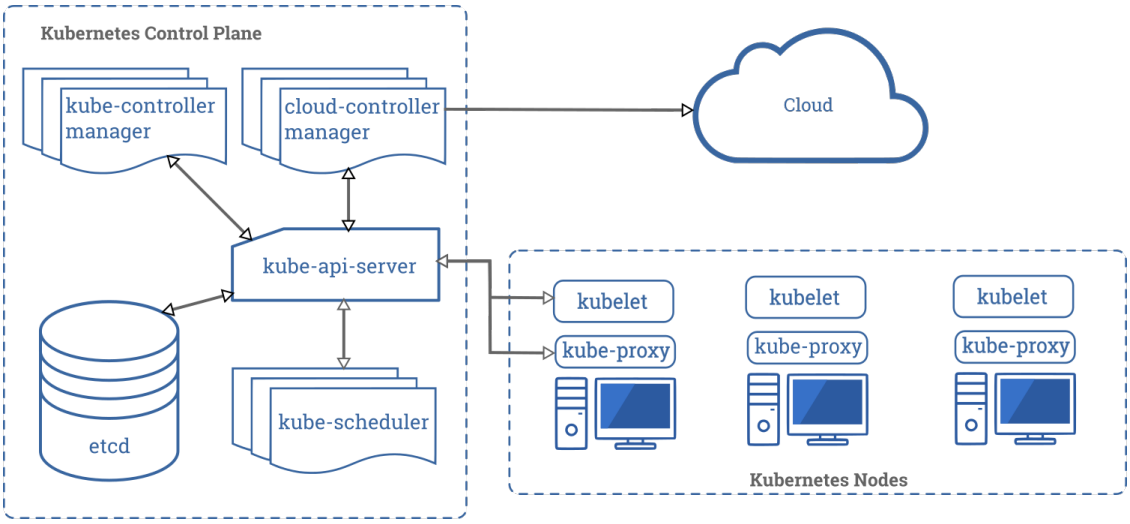


Figura 2.9: Architettura kubernets ad alto livello (fonte : [16])

principale, e da numerosi nodi worker che eseguono le applicazioni in modo distribuito. Solitamente le applicazioni sono inviate al piano di controllo, che le istanzia automaticamente sui nodi worker in base alle policy configurate e al template fornito. I componenti del piano di controllo possono risiedere su un singolo nodo principale o possono essere divisi su più nodi per aumentare la resilienza.
I componenti del piano di controllo sono:

- *l'API server:* implementato con una RESTful interface, che fornisce un punto di ingresso al cluster.Il servizio API viene utilizzato come proxy per esporre i servizi eseguiti all'interno del cluster a client esterni;
- *lo scheduler:* assegna un nodo worker a ciascun componente dell'applicazione;
- *ETCD:* un sistema di archiviazione distribuito per chiave-valore, utilizzato per coordinare le risorse e condividere i dati di configurazione del cluster;
- *il controller manager:* un processo che combina e coordina diversi controller come il replication controller, il controller di nodo, il namespace o il deployment controller.

A questo punto intuiamo che kubernetes funziona con un'architettura distribuita su diversi nodi, i quali devono essere in grado di comunicare tra loro a più livelli. Per le ragioni sopra esposte, possiamo introdurre una nuova importantissima componente

la *Container Network Interface* (CNI).
La CNI fornisce una generica rete, plugin-based, che configura la network-interface dei container e dei nodi all'interno della rete kubernetes; sostanzialmente offre supporto a tutta la catena di forwarding del traffico e comunicazione nella rete dandoci la possibilità di istanziare funzioni di rete e implementarne di nuove tramite eBPF.

Anche Kubernetes verrà utilizzato all'interno della soluzione proposta nel documento; scelto perchè largamente adottato nell'ambito di servizi cloud-based e come gestore di funzionalità di rete (ad esempio nelle reti 5g), offre grande flessibilità, potenza e possibilità di miglioramento grazie al grande entusiasmo della community (viene migliorato ogni giorno anche grazie alla partecipazione delle grandi aziende come Intel[17], che propongono versioni sempre più ottimizzate di kubernets e plugin CNI).

Capitolo 3

Approccio Intent-Based

Questo capitolo tratterà ed approfondirà il concetto di intento insieme alle nozioni ad esso correlate e gli scenari in cui può inserirsi; particolarmente importante perchè analizza i principi e i concetti su cui si basa l'intero lavoro.

3.1 Panoramica

Molti operatori del settore della telecomunicazione, con l'avanzare degli anni, si sono trovati di fronte ad un livello tecnologico sempre crescente che li ha spinti ad acquistare risorse sempre più performanti.
La continua innovazione, e la complessità che ne deriva, crea la necessità di un cambio di paradigma nella gestione dei diversi apparati di rete che sfocia nella volontà di costruire un livello di astrazione, sovrastante la rete, che ha lo scopo di far interagire correttamente diverse risorse, con configurazioni eterogenee, su infrastrutture differenti.
Il problema della complessità, unito all'approccio di gestione autonoma delle reti, apre la strada alla sperimentazione di soluzioni di infrastrutture di rete basate su intenti, già agli inizi dell'anno 2000 [18].

Il concetto di intento, in linea di massima, è una diciarazione arbitraria degli obiettivi che un amministratore di una rete vuole raggiungere, specificati in un linguaggio il più possibile vicino a quello umano. Avere un sistema basato su intenti semplifica notevolmente sia la gestione che l'efficenza delle reti su cui si opera, in modo da rendere l'intervento umano perssochè nullo.
Naturalmente definire un linguaggio per la dichiarazione e l'interpretazione degli intenti non è semplice, ma i progressi nei sistemi di Natural Language Understanding (NLU[1]) insieme agli algoritmi basati su reti neurali, come il Bidirectional Encoder Representations from Transformers (BERT[2]), ci hanno consentito di acquisire le conoscenze per comprendere le richieste fatte dagli utenti, precedentemente espresse in un linguaggio formale, ora espresse in un linguaggio naturale (ad esempio in

[1] https://academic.oup.com/jamia/article/18/5/544/829676?login=false

[2] https://arxiv.org/abs/2103.11943

inglese), favorendo l'estrazione della semantica delle frasi in modo da riuscire ad elaborarla tramite servizi automatizzati.
Grazie a questi studi il concetto di rete autonoma gestita ad intenti diventa sempre più affascinante e promettente.

L'integrazione della tecnologia basata su intenti nei sistemi di gestione esistenti, spinge la proposta di un'architettura organizzata in tre strati[1], illustrata in Figura 3.1.

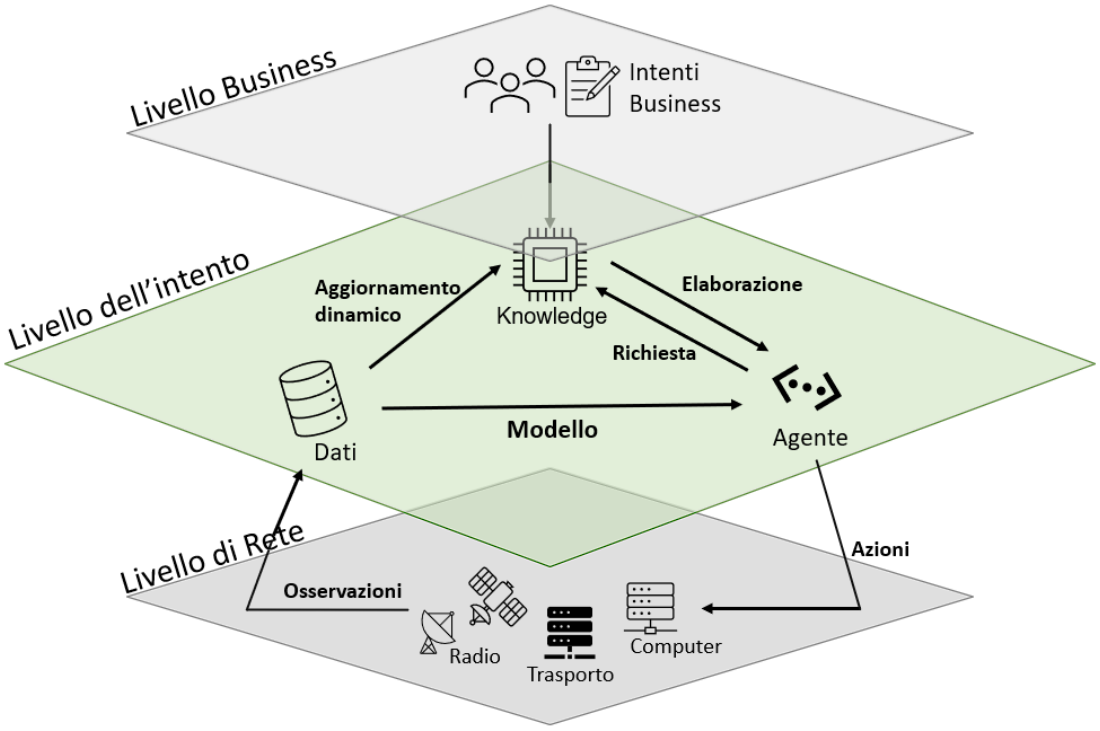


Figura 3.1: Intenti architettura a tre livelli[3]

L'obiettivo è quello di inserire uno strato intermedio definito *Livello dell'Intento*, tra *livello Business e Rete*, che non si limita ad eseguire una sequenza di azioni alla cieca, bensì ad elaborare le richieste del *Livello Business*, imparare da esse, e pianificare azioni di configurazione sul *Livello di Rete*. Naturalmente lo schema non è monodirezionale ma bidirezionale, creando un canale di feedback che aiuta l'utente a comprendere le decisioni dell'*livello dell'intento* e a correggere quest'ultime in caso di errore.

Andiamo ad analizzare nel dettaglio quali sono le componenti fondamentali dell'Intent Layer (Figura 3.1) e come queste interaggiscono con le altre risorse:

- **Knowledge**: gestisce l'astrazione degli intenti, riesce ad elaborarli e a prendere delle decisioni, costruendo relazioni tra diversi oggetti nella rete, per poi comunicarle all'Agente;

[3]Il materiale utilizzato per la creazione della Figura può essere reperito in [1]

- **Agente**: fornisce un'interfaccia agli oggetti di rete (del Livello di Rete) su cui esegue azioni dopo aver valutato gli intenti ricevuti dal Knowledge. L'Agente non ha intelligenza e, quando è richiesta un'azione, invia richieste al Knowledge per l'elaborazione delle decisioni;

- **Dati**: mantiene traccia degli oggetti di rete, ne salva la topologia ed è usata per lo storage permanente delle informazioni. Il suo lavoro principale è quello di comunicare gli aggiornamenti di rete al Knowledge in modo dinamico, ovvero quando si verifica un cambiamento della topologia (anche se riguarda un solo oggetto). Per finire, fornisce il modello della topologia di rete, sempre aggiornato, anche all'Agente.

L'architettura proposta offre diversi vantaggi in termini di scalabilità ed efficienza, ad esempio fornisce una veloce integrazione dello strato di gestione relativo agli intenti nei sistemi moderni, semplificazioni delle configurazioni di rete e di conseguenza dell'addestramento del personale, alta possibilità di standardizzazione dando l'opportunità a diversi vendor di fornire prodotti efficienti ed efficaci. Anche nella proposta di questo lavoro si utilizza una soluzione basata sul Livello dell'Intento, questo permette all'utente di generare velocemente configurazioni valide per più gestori di rete.

3.2 Introduzione agli Intenti

Nel paragrafo precedente 3.1 sono stati citati i concetti di intento, rete autonoma e rete basata su intenti, in questa sezione andremo ad approfondire questi concetti per poi introdurre le tecnologie e le scelte intraprese per completare questo lavoro.

3.2.1 Reti autonome

Per iniziare a parlare di *Intento* dobbiamo innanzitutto definire cos'è una *Rete Autonoma*, dal momento che il termine è stato introdotto per la prima volta proprio nel contesto delle Reti Autonome, dove è definito come: "una politica astratta e di alto livello utilizzata per gestire una rete" (secondo l'RFC-7575[19]).

Le *Reti Autonome* vengono presentate da IBM[4] nel 2001 [19] e si basano sul concetto fondamentale di eliminare dal ciclo di controllo della rete qualsiasi sistema o interazione esterni, mirando ad una rete che possegga capacità di auto-ottimizzazione, auto-configurazione e auto-riparazione. Queste capacità forniscono molti vantaggi, ad esempio l'auto-configurazione della rete permette di configurare e riconfigurare i propri apparati velocemente e in modo automatico, basandosi su metadati, su conoscenza già in possesso del sistema o sugli "intenti", che in questo caso sono visti come un tipo specifico di politica, fornita da un utente, per definire una guida alla Rete Autonoma che altrimenti opererebbe senza pianificazione.

[4]International Business Machines corporation

Particolormente rilevante è anche la capacità di auto-ottimizzazione, la quale permette alla rete di gestire la mole di feedback che riceve per migliorare, se possibile, le performance; ovvero, misurando la deviazione del comportamento atteso da quello ideale si può definire un piano d'azione per perfezionare l'utilizzo delle risorse e, se non per migliorare, quantomeno mitigare il calo di prestazioni. Ovviamente le capacità di auto-ottimizzazione consentono di ottenere un'alta qualità del servizio (QoS) fondamentale per i core business aziendali.
Un'altra proprietà, non meno importante e sicuramente centrale rispetto alle altre, coincide con la capacità di auto-riparazione; questa capacità permette alla rete di reagire in caso di guasti inattesi, possibilmente nel minor tempo possibile, in modo da garantire la continuità del servizio.
Per concludere, legato strettamente al concetto di auto-riparazione c'è quello di auto-protezione, quest'ultimo gestisce casi particolari di guasto dei sistemi, ovvero quei guasti dovuti a cause esterne, spesso dolose, come ad esempio attchi DoS[5].

La Rete Autonoma, di fatto, mira a condurre verso reti che sono fondamentalmente più semplici da gestire e utilizzare, richiedendo solo un intervento minimo esterno, idealmente nullo. Le reti però, anche quando sono autonome, non sono chiaroveggenti e non hanno modo di conoscere automaticamente particolari obiettivi operativi o istanze dei servizi di networking da supportare, in altre parole, non sanno né l'intenzione del fornitore di rete, né lo scopo della loro esistenza.
Per la definizione delle direttive minime di rete si permette all'amministratore di interfacciarsi con protocolli di alto livello, nei quali è possibile dichiarare le configurazioni senza scendere nei dettagli; per queste ragioni, nasce l'intento, come politica informale, per specificare gli obiettivi da raggiungere senza cadere nei vecchi e tediosi costrutti.
Gli obiettivi di design ad alto livello delle reti autonome, oltre a quelli sopra citati, sono molteplici e sono indipendenti dal tipo di soluzione implementativa adottata[20].
Un altro obiettivo, ad esempio, è la definizione di un'infrastruttura e politica comune per tutte le funzioni di rete, dal momento che le funzioni di rete autonome attualmente sviluppate, in numero sempre crescente, necessitano ciascuna dei propri protocolli, architetture e sistemi di interfacciamento. Ad occuparsi di questa problematica è l'Internet Engineering Task Force (IETF), cercando di definire un insieme di servizi comuni di controllo e gestione per quanto riguarda la negoziazione, diagnostica, monitoraggio e distribuzione di intenti.

Il concetto di Secure by Default è un obiettivo cardine nella progettazione di reti autonome, questo ha lo scopo di rendere sicure le interazioni fra processi e dispositivi. Di principale importanza, infatti, è regolare l'autenticazione di ogni dispositivo appartenente al dominio di rete di pertinenza, tipicamente ricorrendo all'utilizzo di certificati[6] rilasciati da un'autorità certificante autorizzata. Questa precauzione

[5]Denial of Service è un attacco informatico in cui l'aggressore cerca di impedire agli utenti di accedere alla rete o alle risorse di un sistema, questo può essere anche distribuito e prende il nome di Distributed Denial of Service e si differenzia in diverse categorie. Per maggiori informazioni consultare https://www.cisa.gov/uscert/ncas/tips/ST04-015.

[6]Nella crittografia asimmetrica un certificato digitale è un documento elettronico che attesta l'associazione univoca tra una chiave pubblica e l'identità di un soggetto che dichiara di utilizzarla

permette di mantenere le comunicazioni sicure, senza dover configurare altri parametri per le interazioni, come ad esempio le chiavi per la crittografia simmetrica[7]. Naturalmente è sempre possibile l'interazione tra apparati appartenenti a domini differenti, a patto che condividano un punto nella catena delle autorità di certificazione. Per concludere, le funzioni implementate nelle reti autonome necessitano di un piano di controllo per i nodi al fine di rendere possibile la comunicazione, senza complicazioni di sorta. L'implementazione di questo componente deve prescindere dal tipo di nodo che lo ospita e qualsiasi apparato, che contiene la funzione, deve supportare questo tipo di interfacce.

3.2.2 Intento, Definizioni e Principi

L'Intento, come idea, ha vissuto una profonda evoluzione rispetto a quando è stato introdotto per la prima volta, allargando la sua influenza a tutte le tipologie di rete che necessitano di un input esterno per definire il proprio comportamento, creando, di fatto, il concetto innovativo di "Intent-Based Networks" (IBN) [21], ovvero di rete guidata o basata sugli intenti. Oltre alle reti, il concetto di intento si infiltra e abbraccia anche tutti quei sistemi che vogliono adottarlo come politica per definire i propri obiettivi operativi, ragion per cui nasce anche il concetto di "Intent-Based Systems" (IBS) [21], analogo al precedente.

Come specificato nella Sezione 3.2.1, l'intento era visto come un particolare tipo di politica inserita dall'amministratore di rete al fine di fornire una direzione operativa alle reti autonome. Ad oggi, si definisce il concetto di intento con un'accezione più ampia e non limitata alla semplice analogia nozionistica presente in letteratura. Nello specifico, l'intento assume una nuova definizione, ovvero: "*l'intento rappresenta una dichiarazione sequenziale di obiettivi operativi, che la rete deve raggiungere, e risultati, che la rete dovrebbe fornire, senza specificare come ottenerli*"[21].
Essendo una definizione puramente dichiarativa, l'intento deve soddisfare contemporaneamente alcuni concetti, quali:

- *astrazione dei dati*: consente una separazione dei compiti che permette agli utenti di non dover occuparsi delle configurazioni di basso livello;
- *astrazione funzionale e dalla logica di controllo*: consente all'utente di disinteressarsi del procedimento necessario al raggiungimento di uno specifico scenario. Ciò che viene specificato nell'intento è solo il risultato desiderato, sarà l'IBS a definire una linea d'azione (ad esempio, utilizzando un algoritmo o applicando una serie di regole derivate dall'intento) per ottenere il risultato richiesto.

Un'altra possibile definizione è fornita da Open Networking Foundation (ONF) [22], che vede l'intento come un'interfaccia dichiarativa fornita da un controllore.

---

nell'ambito delle procedure di cifratura asimmetrica e/o autenticazione tramite firma digitale.

[7]Con crittografia simmetrica si intende una tecnica di cifratura. Rappresenta un metodo per cifrare testo in chiaro dove la chiave di codifica coincide con quella di decodifica.

Questa interfaccia prevede la presenza di funzioni centralizzate che siano in grado di consentire un'interazione con l'utente, tramite la ricezione ed elaborazione di intenti. Il processo di elaborazione prevede che gli intenti vengano trasformati in regole di basso livello e orchestrate lungo tutta la rete.
Quella fornita da ONF è, sicuramente, una definizione diversa, semplicistica in un certo senso, che poggia più su basi pratiche che teoriche, fornendo una visione parziale delle potenzialità della tecnologia se paragonata a quella dell'IETF.

Per fare un pò di chiarezza, si riportano alcuni esempi di cosa potrebbe essere classificato come un vero intento e di cosa potrebbe assomigliarci senza però esserlo effettivamente.

Esempi di intenti:

- "Devia il traffico generato da una sorgente verso una destinazione evitando il passaggio attraverso una determinata area geografica";
- "Massimizzare l'utilizzo della rete anche se ciò significa rinunciare alla qualità del servizio fin quando i livelli di servizio non si deteriorino del 20 % o più rispetto alla media storica";
- "Il servizio VPN nella rete deve sempre fornire percorsi sicuri".

Esempi di dichiarazioni simili ad un intento:

- "Configura l'interfaccia di rete del nodo A con l'indirizzo IP 191.15.23.1";
- "Quando un collegamento raggiunge una saturazione del 20 % emetti una notifica";
- "Configura un web application firewall sull'applicazione A del nodo B che possa scartare il traffico proveniente dall'india".

Gli esempi del secondo blocco non vengono classificati come intenti poichè violano i principi di astrazione sopra riportati; di fatto viene specificato, in ogni pseudo intento, come le proprietà fisice della rete debbano essere configurate, concentrandosi in maggior misura nello specificare come un'azione debba essere portata a termine piuttosto che specificare quale sia l'obiettivo globale. Tuttavia, nell'ultimo esempio del secondo elenco, viene riportato uno pseudo intento che rispecchia l'approccio utilizzato in questo lavoro. L'intento preso in esame, seppur riportato nell'elenco degli utilizzi impropri del linguaggio dichiarativo, coincide perfettamente con il focus del progetto che non è quello di gestire una rete autonoma e tutti i suoi servizi, bensì quello di istanziare servizi di sicurezza con dichiarazioni ad alto livello. Di fatto, la dichiarazione di questo tipo di intenti è coerente con l'utilizzo del framework che abbiamo selezionato, il quale permette di costruire una topologia virtuale tramite un linguaggio dichiarativo riuscendo a rispettare le esigenze progettuali.

Definito cos'è un intento è doveroso annoverare i due principi su cui questo si basa; tuttavia è opportuno includere tutti i concetti affini a quello di intento, emersi anche in precedenza durante la sua trattazione, per effettuare un confronto ed attuare una distinzione netta che ne favorisca la comprensione:

- una **Politica** si identifica in una serie di regole, tipicamente modellate attorno a una variazione di eventi o condizioni, utilizzate per esprimere semplici anelli di controllo che possono essere attuati da dispositivi senza richiedere un intervento esterno. Le politiche consentono agli utenti di definire cosa fare in quali circostanze, senza specificare il risultato desiderato;

- un **Modello di servizio** è un modello di dati utilizzato per descrivere le istanze dei servizi forniti dalla rete ai clienti. Mantenere traccia di come mappare il servizio rispetto alle risorse di rete, di come salvare il legame tra gli oggetti di alto livello con quelli di più basso livello è di fondamentale importanza. Per istanziare un modello di servizio è richiesta l'orchestrazione da parte di un sistema esterno, infatti, la logica per gestire e mappare il modello di servizio su risorse sottostanti non è inclusa nel modello stesso.

Le analogie tra intento, politica e modello di servizio si palesano soprattutto nell'utilizzo di un'astrazione ad alto livello per rappresentare la rete e nell'indipendenza concettuale rispetto al singolo dispositivo. A grandi linee, tutti i concetti tentano di fornire una gestione della rete snella e globale, piuttosto che concentrarsi sulla configurazione manuale di ogni dispositivo.
Per quanto riguarda le differenze rispetto al modello di servizio, gli intenti sono indipendenti dai modelli di basso livello per l'istanziazione dei servizi, inoltre, non dovrebbero aver bisogno di un orchestratore per gestire il servizio in ciascun apparato.
In linea teorica l'intento non necessita di essere orchestrato o suddiviso da un sistema centralizzato a livello superiore, ma dovrebbe essere distribuito a macchia d'olio, cioè consegnato ad ogni singolo dispositivo appartenente alla rete, per poi essere elaborato e processato utilizzando una combinazione di algoritmi distribuiti e astrazioni locali, in modo autonomo, dai singoli apparati. Tuttavia, nella pratica, risulta molto difficile adottare una simile strategia, sia nelle reti virtuali che nelle reti reali, infatti la maggior parte degli apparati di rete, che sono *intent-agnostic*, tendono ad avere un assegnazione esigua di risorse in modo da ottimizzare il rapporto funzinalità prestazione, ragion per cui, non venendo incluso l'overhead di elaborazione degli intenti, si sono adotatte politiche di orchestrazione centralizzate. Confrontando, invece, il concetto di intento con quello di politica, vedicamo l'inutilizzo di catene operative che, una volta scatenate, attuano una serie di azioni per cambiare lo stato della rete in maniera automatica e senza rispettare obiettivi operativi più importanti.

Possiamo procedere ad analizzare i principi fondamentali [21] su cui i sistemi che operano sfruttando il paradigma di intento devono rispettare:

- *Single Source of Truth (SSoT)*: è una componente essenziale per un sistema basato su intenti, in sostanza si occupa di modellare e validare i dati sugli intenti tramite l'aggregazione di informazioni estratte da modelli preconfigurati e ritenuti validi. L'SSoT, in un sistema di gestione basato su intenti, coincide con l'insieme degli intenti ritenuti validi. Grazie a questo controllo si può effettuare un confronto tra lo stato atteso e quello effettivo in modo da poter operare azioni correttive;

- *One-touch but not one-shot*: questo principio si riferisce al modo in cui l'intento viene espresso, accettato ed elaborato dal sistema.
  Esistono sostanzialmente due approcci per la definizione di un intento, il primo definito "zero-touch" sarebbe in grado di riconoscere l'intento in qualsieasi formato venga specificato, il secondo adotta una politica più restrittiva, imponendo una forma a cui l'intento dovrebbe attenersi per essere riconosciuto. Il fulcro del concetto, però, è che indipendentemente dal sistema che si utilizza, la comprensione ed esezucione dell'intento non deve essere istantanea, ma un percorso. L'intento deve essere raffinato prima di essere accettato, per poi eseguire le azioni adeguate al raggiungimento dell'obiettivo specificato. Quindi, in definitiva, un intento non deve essere processato in un colpo singolo, ma elaborato a diversi livelli di astrazione.

- *autonomia e supervisione*: questo principio impone al sistema di raggiungere il giusto grado di autonomia eseguendo i propri compiti senza alcun intervento esterno, inoltre garantisce supervisione, ovvero tramite un sistema di monitoraggio si è in grado di fornire feedback utile all'utente;

- *apprendimento*: secondo questo principio un sistema intent-based deve essere in grado di instanziare un meccanismo di apprendimento, in modo da poter soddisfare sempre gli obiettivi richiesti. L'apprendimento avviene grazie all'interazione con l'operatore che,in caso di errore da parte della rete, attua modifiche opportune "insegnando" alla rete stessa quale sia lo stato giusto da raggiungere, dato l'intento inserito;

- *esposizione delle capacità*: secondo questo principio un sistema deve essere in grado di comporre e scomporre gli intenti, mappando gli obiettivi attesi su determinate azioni o capacità che la rete deve eseguire;

- *astrazione*: questo principio, già trattato nei paragrafi precedenti, presuppone che l'utente non deve preoccuparsi di quali siano le configurazioni del sistema e come queste vengono applicate, ma solo di definirle in modo astratto e dichiarativo.

3.3 Intent-based networking (IBN)

In questa sezione analizzeremo gli aspetti fondamentali ed architetturali delle reti e dei sistemi basati su intenti; infine faremo una breve carrellata dell'evoluzione di questa tecnologia.

In precedenza abbiamo introdotto i concetti di intent-Based Networks e intent-Based Systems seza specificarne peculiarità, differeze e storia.
Quado si parla di IBN ci si riferisce ad una rete gestita da intenti, con la capacità di riconoscerli, elaborarli ed adattare il proprio comportamento, in modo dinamico, per modificare lo stato della rete verso l'obiettivo specificato, senza richiedere interventi esterni. A differenza delle IBN, gli IBS si riferiscono, in senso lato, ad un sistema che permette l'amministrazione di una rete attraverso gli intenti, utilizzando un'interfaccia capace di interaggire sia con l'utente che con la rete per ottenere

lo stato desiderato.
La differenza tra IBN e IBS risulta molto sottile, infatti, seppur entrambe vogliono gestire una rete di comunicazione attraverso gli intenti, nel primo caso si incorpora la comprensione dell'intento direttamente nei dispositivi di rete, nel secondo caso abbiamo un'interfaccia che si occupa di tutta la gestione.
Oggigiorno molte aziende importanti hanno approfondito il concetto di IBN fornendo le più disparate definizioni, ad esempio Gartner[8] definisce una IBN come un software che aiuta a pianificare, progettare e gestire reti, in grado di migliorare la disponibilità e l'agilità della rete stessa. Nello specifico, si inquadrano quattro componenti chiave che un IBN deve fornire[23]:

- traduzione e validazione dell'intento ;
- implementazione degli obiettivi in automatico;
- Consapevolezza dello stato di rete;
- Garanzia del risultato e ottimizzazione dinamica.

Vmware[9], invece, definisce una IBN come un concetto tecnologico emergente, che mira ad applicare un livello più profondo di intelligenza per sostituire i processi manuali di configurazione delle reti e di reazione ai problemi di rete[24]. Infine, si impegnano anche colossi come Cisco[10], tramite pubblicazioni molto dettagliate [25], definendo l'IBN come una rete con la capacità di colmare il gap tra l'azienda e i fornitori di servizio IT, catturando gli intenti e cercando di allineare, continuamente, lo stato dell'infrastruttura agli obiettivi aziendali riguardanti la sicurezza, i servizi e i processi produttivi.

3.3.1 IBN struttura e funzionalità

Le IBN, come specificato più volte, aderiscono strettamente al ciclo di vita dell'intento, come riportato dall'IETF [21], in particolare l'intento, una volta definito, può subire alterazioni, interruzioni e pianificazioni secondo uno schema ben definito.
Per comprendere al meglio le funzioni delle reti basate sugli intenti, facciamo riferimento alla Figura 3.2 che le suddivide in due piani funzionali orizzontali e tre spazi verticali. La suddivisione orizzontale rispecchia sia il concetto di gestione/creazione degli intenti (fulfillment), sia il concetto di verifica della gestione degli intenti (assurance); invece, la suddivisione degli spazi verticali coincide con le interazioni

[8]Società per azioni multinazionale che si occupa di consulenza strategica, ricerca e analisi nel campo della tecnologia dell'informazione, per ulteriori approfondimenti visitare il sito https://www.gartner.com/

[9]Società per azioni, sussidiaria di Dell Technologies, si occupa di sviluppare software per la realizzazione soprattutto per sistemi X86, per ulteriori informazioni consultare il sito https://www.vmware.com/it.html

[10]Cisco Systems Inc., nota semplicemente come Cisco, è una azienda multinazionale specializzata nella fornitura di apparati di networking, per ulteriori informazioni consultare il sito https://www.cisco.com/c/it_it/index.html

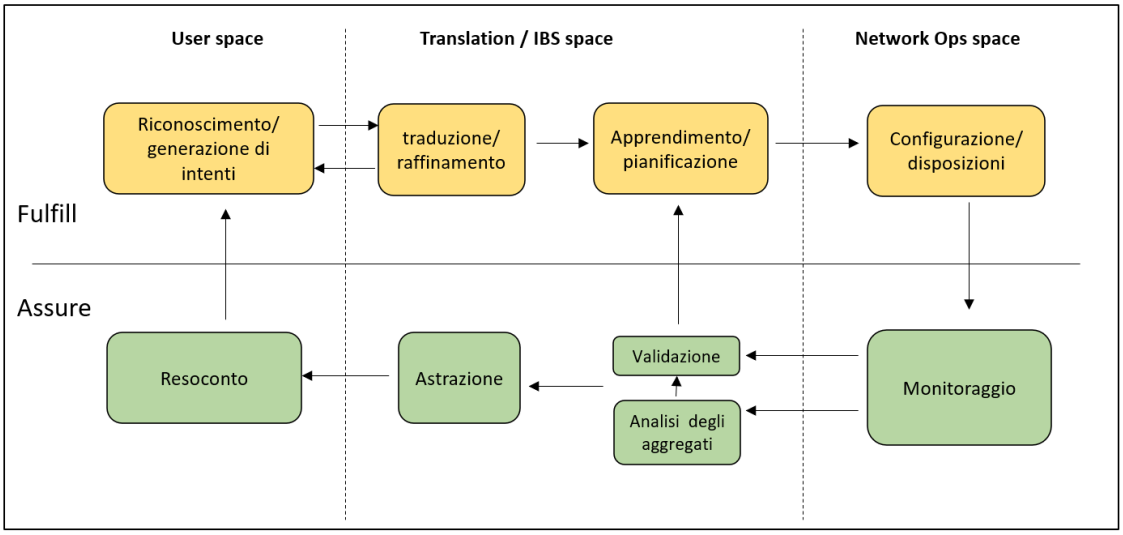


Figura 3.2: Ciclo di vita dell'intento[11]

tra i diversi attori coinvolti.
Per semplicità, l'analisi del ciclo di vita considera la suddivisione funzionale orizzontale come principale in modo da comprendere al meglio come il carico viene redistribuito.

La prima sezione orizzontale che incontriamo, ovvero la *gestione/creazione degli intenti (fulfillment)*, fornisce le funzioni fondamentali che permettono all'utente di interfacciarsi con il sistema, in modo da generare nuovi intenti per poi garantire che l'elaborazione, di questi ultimi, porti ad un cambiamento di stato della rete in linea con la volontà dell'utente stesso. Ancora, include tutte le funzionalità che permettono di ottimizzare la resa del risultato e di gestire, nonchè collegare, tutte le astrazioni di alto livello verso quelle di più basso livello. Le funzioni che operano in questo campo sono:

- *Riconoscimento/generazione di intenti*: questo tipo di funzioni sono quelle che consentono l'interazione tra il sistema e l'utente, forniscono tutti gli strumenti per definire e raffinare un intento senza dover rispettare una sintassi precisa. La funzione consente al sistema di guidare l'utente verso una creazione precisa e chiara dell'intento che sarà poi possibile processare. In realtà la crazione dell'intento è una parte strettamente legata all'utente, di fatto è riporto come attore principale della funzione nella suddivisione verticale;

- *Traduzione/raffinamento*: questo tipo di funzioni si occupano di definire un insieme di operazioni da applicare sulla rete, a fronte di un particolare tipo di intento catturato, che possano modificare le configurazioni degli apparati a basso livello. Sostanzialmente, una volta catturato un intento, in base al linguaggio che si sta utilizzando, se ne effettua un'analisi semantica cercando di mappare i concetti estratti su un linguaggio formale più semplice. La traduzione verso un linguaggio formale ha un duplice scopo, il primo è quello di

[11]Il materiale utilizzato per la creazione della Figura può essere reperito in [21]

verificare che l'obiettivo sia stato compresso correttamente, tramite appositi meccanismi, e il secondo è quello di creare una base per una pianificazione ottimale basandosi su azioni tendenzialmente più semplici da eseguire e raggiungere;

- *Apprendimento/pianificazione*: questo tipo di funzioni lavorano in sinergia perfetta con le funzioni precedenti, infatti ne prendono in ingresso i risultati e cercano di raggiungere l'obiettivo specificato dall'utente nel modo più performante possibile. Queste funzioni devono fornire la capacità di migliorare l'esecuzione delle operazioni, da effettura sulla rete, scegliendo una soluzione in base alle esperienze pregresse di configurazione, riuscendo anche, in alcuni casi, a prevedere lo stato futuro della rete anticipando la risposta del sistema;

- *Configurazione/disposizioni*: queste funzioni sono invece strettamente legate allo spazio di rete, si occupano di attuare le modifiche decise dalle funzioni precedenti.

La seconda sezione che incontriamo, nella suddivisione orizzontale, si riferisce alla *verifica della gestione degli intenti (assurance)*, la quale si occupa di verificare che l'applicazione delle modifiche richieste dall'intento generino il comportamento desiderato. Per effettuare questo controllo si mettono a disposizione diverse funzioni:

- *Monitoraggio*: queste funzioni, specularmente a quelle di configurazione, hanno il compito di raccogliere le informazioni riguardanti lo stato della rete, ovvero tutti gli eventi che si verificano nella rete e lo stato di salute di ogni servizio attivo. I dati raccolti vengono memorizzati e successivamente elaborati;

- *Analisi e validazione degli aggregati*: consideriamo questa pluralità di funzioni come un unico blocco; hanno il compito di analizzare i dati collezionati in fase di monitoraggio, in modo da valutare se le azioni intraprese per soddisfare l'intento siano state correttamente eseguite. Qui, utilizzando il concetto di SSoT, si effettua un confronto tra lo stato attuale e quello atteso, dall'intento ricevuto in input, per poi inviare queste informazioni alle funzioni di apprendimento. Le funzioni di apprendimento potranno basarsi sulla base dati fornitogli per prendere in futuro scelte migliori più velocemente;

- *Astrazione e resoconto*: queste funzioni sono necessarie per fornire all'utente un feedback, ad alto livello, di come la rete stia cambiando il suo stato in riferimento all'intetno inserito. Si rende necessario fornire una visualizzazione dei dati raccolti a basso livello così che l'utente possa intervenire in caso di errore, addestrando e migliorando la rete.

3.3.2 IBN applicazioni e tentativi di standardizzazione

Nel tempo diverse organizzazioni hanno tentato, tramite gruppi di ricerca, di ottenere una standardizzazione dei protocolli riguardanti l'Intent-Based Networking

producendo risultati differenti. Per completezza d'analisi, in questa sotto sezione, andremo ad analizzare quali idee sono state proposte e che risultati sono stati raggiunti.

Il concetto di Intent-Based Network viene menzionato in letteratura, per la prima volta, dal team formato dall'Open Networking Foundation (ONF) nel 2015, rilasciando, per l'occasione, un documento in cui vengono descritti i principi e le idee fondamentali. Nel 2016, lo stesso gruppo, rilascia un nuovo documento [22] in cui si tenta di standardizzare il concetto introdotto l'anno precedente. L'idea prevede la creazione di un gestore di intenti basato su una NorthBound Interface (NBI) che può essere incorporato sia all'interno che all'esterno del controller di rete. Gli sforzi nella standardizzazione continuano ancora oggi, infatti il gruppo di lavoro dell'ONF si è unito a quello di ON.Lab (Gruppo fondatore del sistema di controllo ONOS[12]) per sviluppare lo standard perfetto.

Come l'ONF anche l'ETSI sta facendo ricerca nel campo dei sistemi intent-based con il lavoro del gruppo Zero-touch network and Service Management (ZSM[13]). Questo gruppo ha come obiettivo quello di creare un nuovo standard che abbraccia un argomento molto vasto, ovvero gestire tramite intenti i così detti sistemi automatici; i loro sforzi vengono pubblicati sul piano annuale dell'ETSI[14], ma ad oggi si sono fermati con la loro ultima pubblicazione nel 2020 [26].
Anche l'Internet Engineering Task Force (IETF) propone un progetto con obiettivi simili a quello dell'ETSI, chiamato ANIMA (RFC-8993 [27]), il quale ha lo scopo di sviluppare un modello di rete a gestione autonoma più efficiente, attraverso tecnologie note come YANG[15] e NETCONF[16].

Ovviamente, questo tipo di ricerca non è trattata soltanto da istituti ufficiali, ma anche da aziende private, comunità open-source e ricercatori accademici.
In particolare, il mondo dell'open-source sta analizzando le relazioni tra il concetto di Software Defined Network (SDN) e IBN, tramite gruppi come ONOS e OpenDaylight, introducendo diversi progetti. Due progetti che sono ritenuti sicuramente fondamentali sono il Network Intent Composition (NIC[17]), con l'ambizione di definire una grammatica completa e una sintassi ben definita per descrivere gli intenti, e Intent-based NEtwork MOdeling (NEMO) che sarà approfondito nei paragrafi successivi.

Per concludere, anche la Commissione Europea ha finanziando progetti relativi alla sicurezza, in grado di abbracciare i temi e le problematiche appena descritti. Il progetto FISHY[18] si pone come obiettivo quello di definire uno strato di sicurezza efficace e resiliente sull'intero sistema relativo all'Information and Communications

[12]Controller per soluzioni SDN con funzioni di rete virtualizzate. Si consiglia approfondimento sulla documentazione ufficiale https://wiki.onosproject.org/display/ONOS/ONOS

[13]https://www.etsi.org/technologies/zero-touch-network-service-management

[14]https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=54340

[15]Per approfondire consultare RFC-6020 https://www.rfc-editor.org/rfc/rfc6020.html

[16]Per approfondire consultare RFC-4741 https://www.rfc-editor.org/info/rfc4741

[17]https://test-odl-docs.readthedocs.io/en/latest/developer-guide/network-intent-composition-(nic)-developer-guide.html

[18]https://fishy-project.eu/

Technology (ICT), nonchè di misurare la conformità degli standard di sicurezza cercando, anche, di attuare misure correttive. Per rispettare gli obiettivi proposti dal progetto si studiano diversi domini, tra cui SDN, NFV e IBN.
Uno degli obiettivi di questo lavoro di tesi è quello di fornire un contributo al progetto europeo nei componenti strategici sopra citati.

Capitolo 4

Linguaggi e piattaforme Intent-Based

In questo capitolo vedremo una panoramica dei linguaggi per la definizione degli intenti, analizzandone caratteristiche e potenzialità in riferimento alle tecnologie adottate in questo lavoro.

Prima di procedere, è doveroso fare un breve richiamo ai concetti presentati all'inizio del capitolo precedente (Sezione 3.1) per introdurre formalmente le North Bound Interfaces (NBI), fondamentali nella trattazione degli intenti.
Come già accennato, il Livello dell'Intento è un modulo che si frappone tra il livello applicativo ed il livello di rete, con il compito di elaborare e gestire gli intenti. Questo modulo, di fatto, rispecchia in pieno il concetto di intent-based NBI la cui funzione fondamentale è sicuramente quella di tradurre gli intenti, permettendo di colmare il distacco tra astrazioni di alto livello, con cui gli intenti vengono definiti, e configurazioni di basso livello, in cui sono necessari comandi e azioni concrete per portare la rete in un nuovo stato.
Come definito da [28], la NBI deve essere in grado di ricevere delle espressioni in linguaggio dichiarativo, compilarle, orientandole meno al linguaggio naturale e più al linguaggio strutturato, in modo da poterle elaborare e risolvere applicando il corretto flusso di esecuzione a basso livello. Le funzionalità appena citate vengono implementate in due sotto-moduli definiti Intent Compiler e Intent Solver, in particolare il primo sotto-modulo realizza la validazione della sintassi, assicurandosi che l'intento aderisca alle regole di forma imposte dalla NBI stessa, mentre il secondo processa gli intenti al fine di estrarre informazioni da utilizzare in fase di traduzione.
Secondo questa visione, il controllore della NBI risulta centralizzato e quindi può essere integrato in diversi punti del sistema, consentendo una gestione flessibile attraverso l'utilizzo di linguaggi dichiarativi che possono essere definiti in una moltitudine di formati.

4.1 Linguaggi intent-based

Quando parliamo di linguaggi per la formulazione di intenti non ci riferiamo a caratteristiche specifiche della sintassi, ma più in generale ad un linguaggio dichiarativo

e strutturato, che permetta la definizione chiara di alcune azioni basilari senza scendere nei dettagli implementativi. Detto ciò, comprendiamo che molti linguaggi già esistenti ed utilizzati per altri scopi possono essere utilizzati per formulare gli intenti, infatti linguaggi come Extended Markup Language (XML) e JavaScript Object Notation (JSON) ne sono un importantissimo esempio.
Nei linguaggi appena citati, però, ciò che potrebbe sembrare un vantaggio, ovvero la grande flessibilità, si potrebbe concretizzare in un possibile svantaggio, di fatto essendo linguaggi generici non possiedono regole precise e stringenti. Regole poco strutturate potrebbero portare complicazioni nella definizione della sintassi e nella definizione di eventuali estensioni, del linguaggio stesso, da parte degli sviluppatori, generando problemi anche nella creazione di una comunicaizone standardizzata. Infatti, questi linguaggi dipendono fortemente dall'implementazione della piattaforma su cui vengono utilizzati, perchè si rende necessario definire, a priori, una forma di sintassi accettata che dovrà essere utilizzata in tutte le richieste che vengono effettuate, risultando di fatto poco flessibile (per approfondire è possibile analizzare l'esempio presente nel documento [29]).
Tipicamente i linguaggi per la formulazione di intenti sono spesso creati ad hoc, pensati per essere facili da utilizzare e assimilare, proprio perchè l'utente che dovrà utilizzarli, per definizione dei sistemi intent-based, rappresenta un operatore che non ha competenze specifiche.
Quindi, i linguaggi per la formulazione degli intenti possono essere divisi, sostanzialmente, in due categorie principali [28]:

- *linguaggio imperativo*: categoria che racchiude una tipologia di linguaggi meno elaborati, simili ad un file JSON o XML sia per la struttura che per la sintassi. L'intento è definito in modo schematico, dove, per ogni riga, viene definita una parola chiave e il suo corrispondente valore, in modo che il comando non sia soggetto ad interpretazione;

- *linguaggio dichiarativo*: categoria che rappresenta una tipologia di linguaggi più elaborati ed intuitivi dal punto di vista dell'utente, in quanto espressi in un linguaggio simile a quello naturale. La formulazione di un intento è definita come una frase costituita da un soggetto, un'azione, complemento ed eventualmente dei vincoli.

Per la categoria dei linguaggi imperativi è possibile trovare il linguaggio denominato *Network Intent LanguagE* [30] (Nile), oltre ad i già citati XML e JSON.
Il linguaggio Nile è una rappresentazione intermedia tra linguaggio naturale e linguaggio di basso livello; garantisce una struttura sufficientemente definita da poter essere facilmente elaborato, senza troppe libertà, da sistemi diversi. Oltre alla portabilità, questo linguaggio, possiede una struttura in grado di garantire un agile processo di apprendimento per le funzioni dell'IBN.
Nile, quindi, mira ad ottenere sia una leggibilità elevata, rendendo ogni parola chiave auto-esplicativa, sia un'estendibilità elevata, permettendo ampliamenti della sintassi in modo semplice e veloce.
Per comprendere meglio questo linguaggio possiamo analizzarne la sintassi tramite l'esempio in Figura 4.1, suddiviso in due fasi.
La prima fase, riportata nell'esempio, riguarda la dichiarazione dell'intento tramite

parole chiave "define" ed "intent", assegnandogli il nome "qosIntent", subito dopo viene introdotta la seconda fase, ovvero la dichiarazione degli obiettivi da raggiungere, chiamato corpo dell'intento. Nel corpo dell'intento si parte dalla definizione dei target, "gateway" e "database", introdotti dalle parole chiave "from" e "to", successivamente si procede definendo l'azione desiderata, in questo caso con le parole chiave "add" e "middlebox" si definiscono i servizi da aggiungere. Per concludere, alla fine del corpo dell'intento possono essere definiti dei vincoli, che in questo caso si riferiscono al traffico accettato e alla banda dei collegamenti tra le entità. Un

```
define intent qosIntent:
  from endpoint ('gateway')
  to endpoint ('database')
  for group ('B')
  add middlebox ('firewall'), middlebox ('ids')
  set bandwidth ('min', '100', 'mbps')
  allow traffic ('https')
```

Figura 4.1: Formulazione intento linguaggio Nile[1]

altro linguaggio che può essere analizzato è Merlin [32], per via della sua diffusione ed adozione anche nel mondo delle Software Defined Network. Di solito i linguaggi di gestione delle SDN si concentrano molto sul packet forwarding e ignorano alcuni problemi molto importanti, non riuscendo a fornire API adeguate per una gestione efficiente ed efficace del sistema; Merlin è utilizzato per colmare queste mancanze, integrando una gestione intent-based del sistema.
Per la comprensione del linguaggio Merlin ci baseremo sull'esempio definito in figura 4.2, in modo da analizzarne la sintassi ed ottenere un'idea generale del suo funzionamento.

Metavariables

Locations l	*Values* v
Network functions t	*Variables* x
Header fields f	*Natural numbers* n

Syntax

Bandwidth expressions	e	$::= n \mid x \mid e + e$
Presburger formulas	ϕ	$::= \max(e,n) \mid \min(e,n)$ $\mid \phi_1$ and $\phi_2 \mid \phi_1$ or $\phi_2 \mid !\ \phi_1$
Predicates	p	$::= p_1$ and $p_2 \mid p_1$ or $p_2 \mid !\ p_1$ $\mid f = v \mid$ true $\mid$ false
Path Expressions	a	$::= . \mid l \mid t \mid a\,a \mid a \mid a \mid a^* \mid !\ a$
Statements	s	$::= x : p \rightarrow r$
Policies	pol	$::= [s_1; \ldots; s_n], \phi$

a)

Esempio di policy

```
[ x : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dport = 20) -> .* dpi .* ;
  y : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dport = 21) -> .* ;
  z : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dport = 80) -> .* dpi *. nat .* ],
  max(x + y,50MB/s) and min(z,100MB/s)
```

b)

Figura 4.2: Merlin sintassi astratta ed esempio[2]

[1]Le informazioni utilizzate per l'esempio si rifanno al [31]

[2]Le informazioni utilizzate si riferiscono al [32]

Formalmente, come osservando in Figura 4.2b, Merlin utilizza un linguaggio basato su delle policy, definendo una grammatica (Figura 4.2a) costituita da un insieme di "dichiarazioni" e "formule logiche". Ogni dichiarazione specifica la gestione di un sottoinsieme di traffico, mentre la formula logica esprime un vincolo globale sulla larghezza di banda.
Scendendo nel dettaglio sintattico del linguaggio, una dichiarazione è costituita da una variabile, da un predicato logico e da un'espressione regolare; i predicati logici includono espressioni atomiche nella forma f (*field*) = v (*value*), nell'esempio l'espressione confronta l'indirizzo ip sorgente con un indirizzo noto (value), ma è possibilie anche esprimere diversi tipi di espressioni, come quelle booleane di and, or e la loro negazione (!).
Le espressioni regolari sono introdotte dal simbolo "→" e possono corrispondere a funzioni o locazioni di rete. Come per le espressioni regolari POSIX, il simbolo "." definisce un match con una locazione arbitraria di un elemento di rete, scelta dal compilatore. Per concludere la spiegazione della sitassi, inoltre, è possibile definire dei vincoli sia sulla banda utilizzata, tramite le parole chiave "max" e "min", sia sul traffico aggregato, tramite il simbolo di addizione "+". Compresa la sitassi, possiamo finalmente analizzare l'esempio per la definizione di una policy in Figura 4.2b, suddiviso in tre espressioni fondamentali, x, y e z, ed un vincolo. Il primo ed il secondo predicato che incontriamo indicano che il percorso del traffico FTP[3], specificato dalla porta 20 e 21 di comunicazione tra i due indirizzi IP, sia indirizzato verso una funzione di Deep Packet Inspection (DPI[4]), senza aggiungere altri vincoli. Continuando con il terzo predicato, viene aggiunto un vincolo sul percorso del traffico HTTP[5] tra i nodi DPI e NAT[6] e, per concludere, si inserisce una limitazione globale sul traffico aggregato, con porte di destinazione 20 e 21, limitando la banda a 50MB/s, mentre al traffico relativo alla porta 80 viene garantita una banda minima di 100MB/s.

La progettazione dei sistemi intent-based non si limita all'utilizzo di un solo linguaggio o di una sola strategia, bensì può utilizzarne un mix in modo da sfruttare i pregi di ogni tecnologia selezionata.
LUMI [31] è un progetto ambizioso che utilizza proprio un mix di tecnologie, tra cui i linguaggi Merlin e Nile, con lo scopo di consentire ad un operatore di "discutere con la rete"; cioè, LUMI prende come input l'intento di un operatore, espresso in linguaggio naturale, lo elabora in modo da tradurlo in comandi di configurazione ed esegue sulla rete i comandi ottenuti per raggiungere l'obiettivo prefissato.
LUMI si compone di un'architettura modulare, in ordine: *estrazione delle informazioni, assemblaggio dell'intento, conferma dell'intento, compilazione e distribuzione dell'intento*. Questo design permette di rendere, in primis, il progetto plug-and-play

[3]Protocollo standard analizzato nell'RFC-959 https://www.rfc-editor.org/rfc/rfc959

[4]Deep packet inspection, tipo di elaborazione dati che ispeziona in dettaglio i dati inviati su una rete di computer e può intraprendere azioni come l'avviso, il blocco, il reindirizzamento o la registrazione del paccetto.

[5]Protocollo standard analizzato nell'RFC-9114 https://www.rfc-editor.org/rfc/rfc9114

[6]https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/15-mt/nat-15-mt-book/iadnat-addr-consv.html

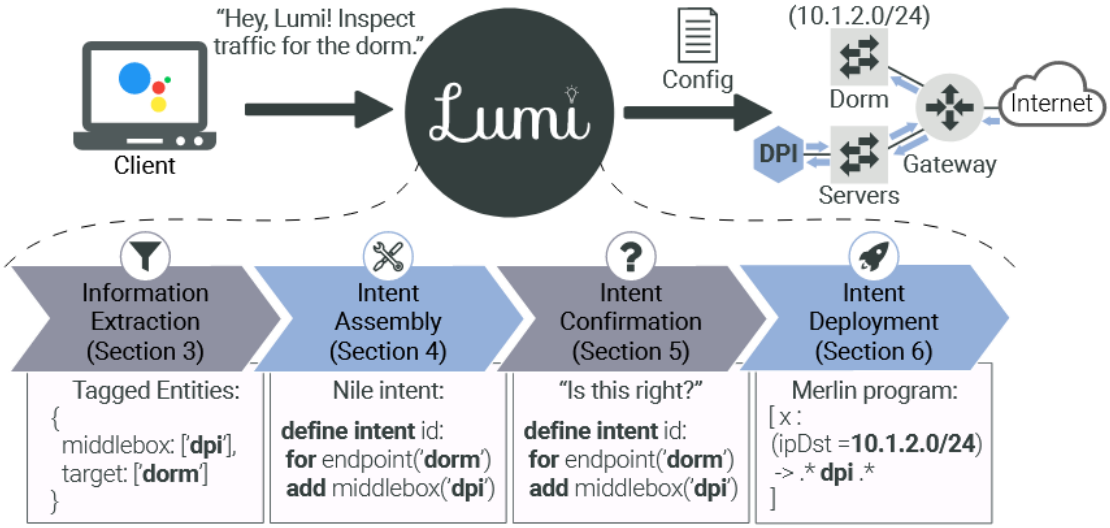


Figura 4.3: LUMI design modulare (fonte: [31])

e, in secondo luogo, di sostituire o creare i moduli all'interno del sistema consentendo di migliorarlo ed aggiornarlo con estrema facilità. Un esempio della composizione dei moduli di LUMI e del suo flusso operativo si evince dalla Figura 4.3, dove è chiaramente visibile l'utilizzo del linguaggio nile per tradurre le informazioni ricevute dall'operatore, con possibilità di autoapprendimento tramite feedback realtime, e dell'utilizzo di merlin per generare la lista delle configurazioni generiche che verranno applicate alla rete SDN.

Per quanto riguarda la seconda categoria di linguaggi analizzeremo Flowlog [33] e NEMO, quest'ultimo sarà analizzato dettagliatamente nelle prossime sezioni. Flowlog, come Merlin, è un linguaggio utilizzato per la gestione delle SDN, composto da una serie di regole, che manipolano il comportamento della rete, e da una serie di vincoli, che regolano le tabelle di stato degli script preconfigurati nel linguaggio. Detto ciò, il linguaggio risulta essere particolarmente complesso da analizzare per via della presenza delle tabelle preconfigurate[7], nonostante questo, ricorrendo all'esempio in Figura 4.4, possiamo definire a le sue caratteristiche principali ed il suo comportamento.

Ogni intento definito con Flowlog contiene un "ruleset[8]" dichiarativo, che regola il comportamento del controllore in riferimento ad una serie di "dichiarazioni", che possono coincidere con le tabelle di stato del programma, con le interfacce per le tabelle di *INCOMING* e *OUTCOMING* o, più semplicemente, con un evento generico. Nella Figura 4.4b si può vedere una dichiarazione di un rulset dove, sostanzialmente, ogni regola processa un evento per volta prendendolo da una tabella di INCOMING, processandolo ed inserendolo in una tabella di OUTCOMING. Di

[7]Con il termine *tabelle preconfigurate* si intendono tutte quelle tabelle che servono al linguaggio FlowLog per confrontare gli eventi a cui deve reagire, valutare lo stato del sistema e confrontare le informazioni con lo storico di quelle già accumulate.

[8]Per ruleset, in questo caso, si intende l'insieme di regole definite in FlowLog a cui si deve far riferimento nel momento in cui si vogliono definire delle azioni.

Syntax

```
block ::= ON <id> ( <id> ) [WHERE rformula]
          : rules
rules ::= rule | rule rules
rule  ::= do_act | ins_act | del_act
do_act  ::= DO <id> ( termlist )
            [WHERE rformula] ;
ins_act ::= INSERT ( termlist )
            INTO <id> [WHERE rformula] ;
del_act ::= DELETE ( termlist )
            FROM <id> [WHERE rformula] ;
term ::= <num> | <string> | <id> | <id>.<id> | ANY
termlist ::= term | term , termlist
rformula ::= <id> ( termlist ) | term = term |
             NOT rformula | rformula AND rformula |
             rformula OR rformula | ( rformula )
```

a)

Esempio di policy

```
1  ON packet_in(p) WHERE p.dlTyp = 0x1001
2    AND dstSw = p.locSw AND srcSw = p.dlSrc:
3    INSERT (srcSw, dstSw) INTO ucTC;
4    INSERT (sw, dstSw) INTO ucTC
5      WHERE ucTC(sw, srcSw);
6    INSERT (srcSw, sw) INTO ucTC
7      WHERE ucTC(dstSw, sw);
8    INSERT (sw1, sw2) INTO ucTC
9      WHERE ucTC(sw1, srcSw)
10     AND   ucTC(dstSw, sw2);
```

b)

Figura 4.4: Flowlog sintassi e policy d'esempio (fonte: [31])

base ogni regola, scatenata da una condizione specifica, inizia con il comando *ON* ed è seguita dall'entità a cui fa riferimento. Le azioni possibili da eseguire sull'entità sono specificate tramite i comandi *INSERT o DELETE*, mentre per azioni basilari è possibile utilizzare l'opzione *DO* (un esempio potrebbe essere il forward[9]). Infine, è presente un'ulteriore clausola *WHERE* che permette di configurare vincoli aggiuntivi a quelli già definiti con le clausole precedenti.
Compresa la sintassi (Figura 4.4a), analizziamo l'esempio in Figura 4.4b dove si vuole definire un vincolo sulla tabella dei pacchetti ricevuti, per ogni pacchetto che rispetti le caratteristiche specificate nella WHERE, inserendo l'elaborazione finale in una tabella di OUTCOMING. Nel testo di riferimento l'esempio è molto più articolato e le regole si riferiscono allo smistamento del traffico di rete, ma per comprendere meglio il linguaggio è stata effettuata una semplificazione.

Le ricerche in questo campo sono molteplici ed ogni ricerca propone un linguaggio specifico a cui non possiamo dedicare troppo tempo, d'altro canto possiamo far menzione di alcuni linguaggi utilizzati nel contesto degli intenti, particolarmente rilevanti, come:

- *Frenetic*[10]: con il termine Frenetic si fa riferimento ad una famiglia di linguaggi che hanno come obiettivo quello di rendere i linguaggi per lo sviluppo delle reti semplici ed il più possibile esenti da errori. Frenetic rispetta i concetti di *alto livello di astrazione*, *implementazione modulare* e *portabilità* mantenendo una semantica rigorosa. Attualmente sono in sviluppo due linguaggi per la famiglia Frenetic, ovvero Frenetic-OCaml[11], implementato in OCaml, e Pyretic[12], implementato in python. Non mancano progetti che estendono

[9]Per forwarding si intende il trasferimento dei pacchetti da un segmento di rete ad un altro da parte dei nodi in una rete di computer.

[10]*http://www.frenetic-lang.org/*

[11]https://github.com/frenetic-lang/frenetic

[12]http://www.frenetic-lang.org/pyretic/

ed utilizzano questo linguaggio come Ox[13], Kinetic[14] e per finire Merlin già citato in precedenza;

- *Gherkin*[15]: è un software utilizzato per raffinare il linguaggio formale in un linguaggio più semplice e snello, viene utilizzato nel sistema Cucumber[16] per generare un insieme di regole strutturate in grado di essere comprese dal sistema stesso. Gherkin utilizza una serie di parole chiave "speciali" per dare struttura e significato alle specifiche eseguibili; ogni parola chiave è tradotta in una pluralità di lingue parlate così da renderne semplice l' apprendimento e l'applicazione;

- *Cisco ACI language*[17]: è un linguaggio di configurazione creato da Cisco basato su JSON o XML, permette un interazione con l'utente semplice e flessibile.

4.2 Intent-based NEtwork MOdeling language

In questa sezione tratteremo nel dettaglio il linguaggio, creato da Huawei, chiamato Intent-based NEtwork MOdeling, effettuando un'analisi del suo sviluppo, della sua struttura e della sua implementazione nel tempo.

Il termine NEMO, utilizzato più volte nel campo dell'Intent Based Networking, è stato introdotto per la prima volta da Huawei in riferimento ad un linguaggio dichiarativo per una NBI di un controllore SDN. Successivamente, Huawei utilizza lo stesso termine per riferirsi allo sviluppo di un nuovo progetto, il quale consiste nella creazione di una NBI, completa, fondata sul linguaggio IB-NEMO.
L'introduzione da parte di Huawei del linguaggio IB-NEMO risulta essere un tassello fondamentale nel mondo dei linguaggi basati sugli intenti, infatti persino l'IETF, considerando i tempi maturi, analizza il progetto NEMO decidendo di tentare la standardizzazione del linguaggio pubblicando nel 2015 il draft "Intent-Based Nemo Overview" [34]. Nel draft vengono evidenziati i vantaggi che la struttura del linguaggio porta con se, nonchè l'estrema semplicità di utilizzo.
Seguendo l'esempio dell'IETF, anche il mondo dell'open source si muove, in particolare OpenDaylight rilascia la prima versione di un progetto basato sul linguaggio NEMO, chiamato apputo OpenDaylight NEMO[18], che ha lo scopo di gestire una rete SDN tramine intenti.

[13] Piattaforma sviluppata in OpenFlow.

[14] una piattaforma di controllo della rete basata su una macchina a stati.

[15] *https://cucumber.io/docs/gherkin/reference/*

[16] https://cucumber.io/docs/guides/overview/

[17] *https://www.cisco.com/c/en/us/solutions/data-center-virtualization/application-centric-infrastructure/index.html*

[18] https://test-odl-docs.readthedocs.io/en/latest/user-guide/nemo-user-guide.html

4.2.1 Struttura e sintassi

Huawei con il progetto NEMO, basandosi su una visione "application-centric" della rete, mirava ad ottenere un modello applicativo capace di raggiungere con facilità gli obiettivi dichiarati dall'utente, ignorando i dettagli ritenuti non necessari.
Sostanzialmente, il progetto voleva soddisfare la necessità di creare un linguaggio comune per la definizione degli intenti, in grado di implementare un'architettura NBI applicabile a tutti i sistemi. Grazie a questa architettura, si concretizzava la possibilità di comunicare con il sistema di gestione della rete, abbassandone la complessità e alleggerendo il carico sull'intent engine.
Il linguaggio che è stato creato aveva lo scopo di essere semplice e comprensibile da una pluralità di utenti, in modo da renderne elementare l'utilizzo, grazie alla peculiarità di auto esplicazione che la sintassi del linguaggio porta con sé. Per queste ragioni, la sintassi di IB-NEMO si basa sulla regola dell' "80/20[19]" per la definizione delle keyword, secondo la quale l'80% delle applicazioni utilizzano solamente il 20% dei comandi di cui una API è provvista, e applicando una semantica molto simile al linguaggio naturale. Ora, per ovviare alla mancanza di supporto per il restante 20% delle applicazioni, si prevede la possibilità di estendere il linguaggio, in modo semplice ed efficace.
La vera sfida consisteva nel definire quali comandi sarebbero stati necessari per comporre il linguaggio; Huawei per risolvere questa problematica ha utilizzato un approccio non convenzionale, infatti analizzava i casi d'uso ritenuti comuni e rappresentativi del dominio, fornendone un possibile codice prototipale che avrebbe dovuto permettere di configurare un determinato scenario. Dopo una prima analisi e prototipazione, veniva sviluppata una possibile soluzione di test utilizzando lo stesso codice ottenuto in precedenza e valutandone il risultato, il quale se fosse stato congruente con lo stato atteso concedeva la possibilità di iniziare la fase di ottimizzazione, fornendo l'opportunità di snellire al massimo i comandi utilizzati.
Solitamente la catena di casi d'uso, su cui configurare il linguaggio, veniva scelta concordemente con i fornitori di servizio più importanti, i quali erano in grado di fornire anche statistiche di utilizzo sui casi d'uso più comuni o complessi.

Il cuore della sintassi del linguaggio NEMO, costruita con le metodologie sopra indicate, viene raffigurato in Figura 4.5 e la configurazione di un intento può essere riassunta dalla tripletta di entità Oggetto, Operazione, Risultato. Dallo schema si evince che ogni entità non costituisce un singolo elemento, bensì si compone di diversi elementi che permettono di esprimere gli intenti in modo flessibile ed efficace. Per descrivere un intento sono previsti due approcci, il primo utilizza un'entità oggetto accoppiata con un'entità operazione, il secondo, invece, accoppia un'entità oggetto e con un'entità risultato.
Per comprendere meglio la sintassi andiamo ad analizzare nel dettaglio le entità principali:

- *Oggetto* si suddivide in tre elementi che rappresentano le entità fondamentali in una rete:

[19] https://arxiv.org/PS_cache/cond-mat/pdf/0412/0412004v3.pdf

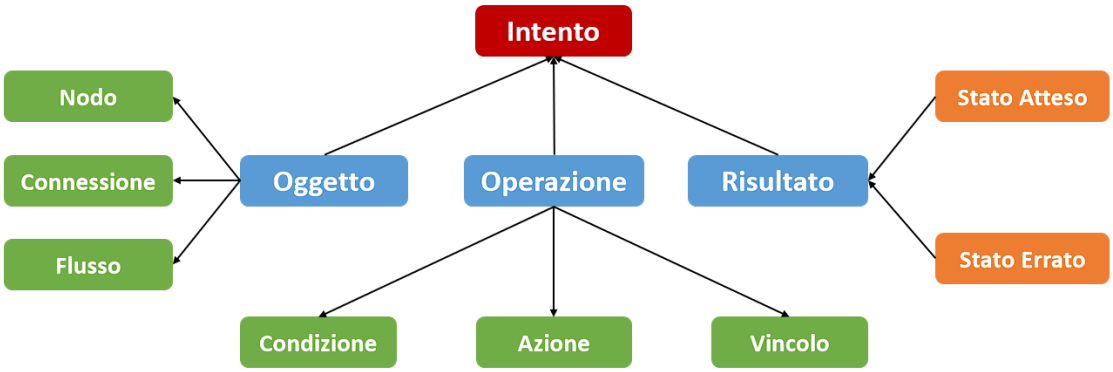


Figura 4.5: Modello della sintassi di IB-NEMO (fonte: [35])

  - *Nodo*: un tipo di risorsa di rete che descrive un oggetto virtuale con la capacità di elaborazione dei pacchetti. Può includere un insieme di elementi di rete, come sottoreti, host, router, ma può includere anche funzioni di servizio di rete come firewall, load balancer, DPI, ecc;
  - *Connessione*: elemento che descrive la risorsa di collegamento virtuale tra le entità nodo. La connessione virtuale ha lo stesso significato di collegamento di rete fisico, che si riferisce alla connessione diretta tra nodo e nodo, tra nodo e utenti;
  - *Flusso*: descrive il traffico presente nella rete, che facilita agli utenti la descrizione dei servizi e l'applicazione di alcune politiche. Il traffico di rete esiste effettivamente nella rete fisica e non può essere creato dal linguaggio (sono utilizzati flussi noti come traffico TCP-IP[20]).

- *Operazione* è un entità per controllare il comportamento di elementi specifici, come le operazioni di flusso, le operazioni sui nodi o le operazioni sulle connessioni. Tutte le operazioni seguono sempre lo stesso schema "When ⟨condizione⟩, do ⟨azione⟩, with ⟨vincolo⟩" e possono essere sempre applicate. In un certo senso, le operazioni hanno un significato simile a quello delle politiche e si suddividono in:
  - *Condizione*: condizione scatenata al verificarsi di un'azione;
  - *Azione*: azione scatenata al verificarsi di una condizione;
  - *Vincolo*: vincolo che viene aggiunto ad una data azione.

- *Risultato* definisce operazioni di confronto e feedback che sono implementate nell'applicazione specifica del linguaggio, non prevede una standardizzazione, infatti segue le regole base del linguaggio, ma definisce due entità che possono essere utilizzate come linee guida:
  - *Stato Atteso*: definisce uno stato che la rete possa raggiungere;
  - *Stato Errato*: definisce lo stato che la rete non deve assolutamente raggiungere.

[20]Per maggiori informazioni consultare le specifiche RFC relative a TCP/IP http://nuovolabs.fauser.edu/~valeria/materiale-didattico/sistemi-quinta/rfc.pdf

```
<NEMO_cmd> ::= <model_definition_cmd> | <resource_access_cmd> |
                          <behavior_cmd> | <transaction_cmd>

<model_definition_cmd> ::= <cfn_definition> | <connection_definition>
                           | <action_definition> | <model_description>

<resource_access_cmd> := <cfn_create> | <cfn_import> |<cfn_update> |
                         <cfn_del> | <connection_create> |
                         <connection_update> | <connection_del> |
                         <serviceflow_create> | <serviceflow_update>
                            | <serviceflow_del>

<behavior_cmd> ::= <query_cmd> | <operation_create> |
                   <operation_update> | <operation_del> |
                   <notification_create> | <notification_update> |
                   <notification_del>

<transaction_cmd> := <transaction_begin> | <transaction _end>

<key_word> ::=  Range | Integer | String | < UNUMBER > | < ID >|
                < TEMPID > | < ETHPREF > | < IPV4PREF > | < DATEVAL >|
                < TIMEVAL > | < FULLTIME > | < ETHADDR > |
                < IPV4ADDR > | < URI > | < UBYTE > | < HEXDIGIT > |
                < YEAR > | < SMONDAY > | < LMONDAY > | < HOUR > |
                < MINUTE > | < SECOND > | < DIGIT > | NodeModel |
                ConnectionModel | ServiceFlowModel | ActionModel |
                Description | Porperty | Node | Connection|
                ServiceFlow | ConnectionPoint | EndNodes | Type |
                Contain | Match | List | Flow | End |
                nodes | connections | flows | operations | at | http |
                https | file | Range | Query | From | Notification |
                Listener | Operation | Target | Priority | Condition |
                Action | Transaction | Commit | CREATE | IMPORT |
                UPDATE | DELETE

<connection_create> ::= CREATE Connection <connection_id>
                        Type <connection_type>
                        EndNodes <node_id>, <node_id>
                        [Property {<property_name>: <value >}];

<connection_del> ::= DELETE Connection <connection_id >;

<node_create> ::= CREATE Node <node_id> Type <node_type>
                  [Contain {<node_id >}]
                  [Property {<property_name>: <value >}];

<node_del> ::= DELETE Node <node_id>;
```

Figura 4.6: Sintassi parziale IB-NEMO (fonte: [36])

In Figura 4.6 viene raffigurata la sintassi parziale del linguaggio NEMO, in

formato BNF[21] (Backus Normal Form), che ci consente di analizzare le principali classi di comandi.
NEMO fornisce cinque classi di comandi, ovvero *definizione di un modello, accesso alle risorse, comportamento, connessioni e transazioni.* Per ogni comando in Figura 4.6, definito come un'espressione regolare, sono riportate l'elenco delle parole chiave, con relativo tipo, che possono essere accettate.
Più in generale, la maggior parte dei comandi ascrivibili tramite NEMO iniziano sempre con una keyword tra CREATE, IMPORT, UPDATE e DELETE, seguito dal complemento oggetto, per specificare il target su cui si opera, e da svariate informazioni supplementari ed opzionali in base alla tipologia dell'elemento a cui si fa riferimento. La tabulazione e la punteggiatura ricoprono un ruolo fondamentale, infatti ogni keyword è separata da un solo spazio o da un segno di punteggiatura ed ogni comando termina con punto e virgola; nello specifico, si può fare riferimento all'espressione regolare per definire i modelli "model_definition_cmd" oppure al prototipo di definizione di un nodo "node_create" (Figura 4.6). Per concludere, il linguaggio è case sensitive così da rendere la definizione degli ID e delle keyword ben distinte, concedendo diversi gradi di libertà anche nella nomenclatura del linguaggio stesso.

Nella categoria dei comandi di accesso alle risorse vengono analizzati, nel particolare, le sequenze di creazione di nodo e di connessione.
Per quanto rigurada la sequenza di creazione di un nodo si notano i vincoli nella definizione del nome in formato ⟨ID⟩, nella definizione del tipo, anch'esso in formato ⟨ID⟩, dove risulta obbligatorio trovare una corrispondenza tra i tipi già definiti nella piattaforma e, per concludere, nella definizione delle proprieta anch'esse già definite all'interno dalla piattaforma di utilizzo.
Invece, per quanto rigurada la sequenza di creazione di una connessione i vincoli sulla definizione del nome, del tipo e delle proprietà sono comuni alla definizione del nodo, l'unica discrepanza si trova nella definizione degli EndNodes, nodi di terminazione, identificati da due stringhe univoche in formato ⟨ID⟩ e già presenti all'interno del sistema.

Per comprendere al meglio la sintassi del linguaggio appena introddotto, possiamo analizzare i seguenti esempi:

- ```
  CREATE Node Waf_Service_One Type waf Property
      application_to_protect: generic, alert_level: higth,
      service_port: 80, active_protection_attack:
      xss-sql-dos-php-java-rce-ip-shell, block_bad_user_agent:
      bot-scanner, block_country_traffic: it, block_domain_name:
      polito.app.it;
  ```

  Nell'esempio si vuole illustrare la creazione di un nodo semplice, specificando tutti gli attributi previsti dal suo modello. L'intento è introdotto dalla keyword CREATE, dal nome assegnatogli, dal tipo di nodo che vogliamo creare e da tutte le sue proprietà. In questo caso il nome del nodo è *Waf_Service_One*,

[21] https://www.techopedia.com/definition/24061/backus-normal-form-bnf
  ```

la tipologia del nodo coincide con *waf* e le sue proprietà sono descritte secondo lo schema "*Property ⟨property_name⟩: ⟨value⟩*". Naturalmente il nodo si basa su un modello predefinito in cui il *Value* delle proprietà è descritto tramite attributi semplici[22];

```
1 CREATE NodeModel vlc_complex; Node emitter Type vlc_vnfd; Node
    receiver Type vlc_vnfd_2; ConnectionPoint mgmt2_tx;
    ConnectionPoint mgmt_trans; Connection data2 Type p2p
    Endnodes emitter.eth0, receiver.eth1; Connection control1
    Type p2p Endnodes emitter.mgmt, mgmt2_tx; Connection control2
    Type p2p Endnodes receiver.mgmt1, mgmt_trans;
2
3 CREATE Node video Type vlc_complex;
```

In questo esempio vogliamo generare la creazione di una pseudo rete tramite la definizione di un nodo complesso. Fondamentalmente il testo è strutturato in due parti, la prima rigurarda la creazione del modello di un nodo complesso e la seconda riguarda l'istanziazione del nodo stesso.
La prima parte dell'intento è introdotta dal comando CREATE, seguito da NodeModel e dal nome che vogliamo assegnargli, successivamente si va a definire come il modello deve essere strutturato. Il modello è composto da due nodi, rispettivamente di tipo vlc_vnfd e vlc_vnfd2, e dalle connessioni create tra i due nodi. Per creare una connessione si utilizza la parola chiave Connection, seguita dal tipo di connessione e dai riferimenti ai rispettivi Endnodes. Nell'esempio vengono create tre connessioni, la prima di comunicazione e le atre due di controllo. Naturalmente è sempre possibile modificare il modello, utilizzando il comando UPDATE e riscrivendone la definizione. Infine, abbiamo la dichiarazione di creazione del nodo complesso sempre introdotta dal comando CREATE.

4.2.2 OpenDaylight NEMO

Per concludere questo capitolo introduciamo la piattaforma di OpenDaylight, appartenente al mondo open source, affermata nel campo delle SDN. Negli ultimi anni, la piattaforma si è interessata al tema dei sistemi intent-based dando alla luce diversi progetti, tra cui OpenDaylight Group Based Policy (GBP)[23], OpenDaylight Network Intent Composition (ODL-NIC)[24] e OpenDaylight Network Modeling: NEMO[37].

[22] NEMO fornisce un insieme di tipologie di dati, tra cui Boolean, Integer, String, Date e UUID (Unique IDentifier standardizzato nell'RFC-4122)

[23] https://docs.opendaylight.org/en/stable-fluorine/user-guide/group-based-policy-user-guide.html

[24] https://docs.opendaylight.org/en/stable-fluorine/user-guide/network-intent-composition-(nic)-user-guide.html

In questa sottosezione andremo ad analizzare il progetto OpenDaylight NEMO, implementazione open source basata sul linguaggio IB-NEMO, che aderisce perfettamente a quanto riportato in precedenza.
Il progetto prevede l'implementazione completa delle entità Oggetto e Operazione, a differenza dell'entità Risultato che appare ancora in stato di sviluppo [38]. Ogni elemento corrisponde ad un tipo, appartenente ad una determinata entità, che possiede svariate proprietà caratterizzanti; tutti i parametri citati devono essere configurati dall'utente prima di interaggire con la piattaforma.

Nella Figura 4.7 viene mostrata la panoramica dell'architettura del progetto NEMO, che contiene le tre componenti principali del NEMO Engine: Physical Network Manager, User Manager e Renderer.
I dati di NEMO sono memorizzati nel repository degli utenti nell'MD-SAL data store, specificati dall'utente nelle applicazioni esterne che comunicano con il NEMO engine attraverso le API RESTful, generate da MD-SAL RESTconf secondo il modello NEMO, e consumate dai componenti User Manager e Renderers.

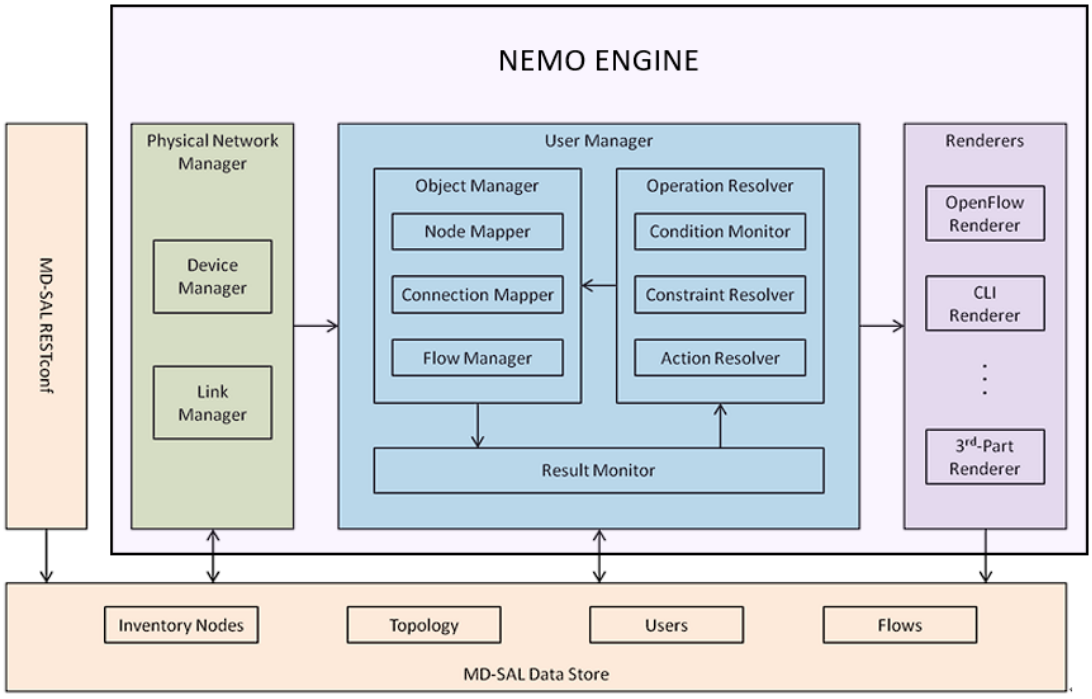


Figura 4.7: Architettura OpenDayligh-NEMO[25]

Analizziamo le componenti dell'architettura nel dettaglio [39]:

- Il *Model-Driven Service Adaptation Layer (MD-SAL)* è un componente middleware estensibile ispirato al message-bus che fornisce funzionalità di messaggistica e di archiviazione dei dati basate sui modelli di dati e di interfaccia definiti dagli sviluppatori delle applicazioni. In questa architettura è utilizzato per regolare lo scambio dei dati e il loro relativo formato. Questa componente verrà analizzata nei successivi capitoli;

[25]L'immagine usa come referenze le informazioni presenti in [39]

- Lo *User Manager* è uno dei componenti principali del NEMO engine e contiene tre sottomoduli, ovvero Object Manager, Operation Resolver e Result Monitor, che si occupano separatamente di oggetti, operazioni e risultati. Lo user manager è principalmente responsabile dell'autenticazione degli utenti e della gestione dei privilegi, nonché del monitoraggio delle modifiche ai dati NEMO e del loro invio ai sottomoduli corrispondenti, in modo da creare un corrispettivo tra Physical Network Manager e dati salvati in MD-SAL data store. Questa componente è in grado di ricevere ed elaborare richieste in due formati differenti, la prima definita *language-style* rappresenta un'intento sottoforma di linguaggio dichiarativo, la seconda, invece, rappresenta un'intento sottoforma di linguaggio imperativo, utilizzando le strutture JSON;

- Il *Renderer* è una componente progettata per far sì che l'architettura NEMO supporti vari elementi di rete virtuale basati su tecnologie sottostanti molto diverse. Gli altri componenti non hanno bisogno di prendere in considerazione i dettagli dell'implementazione sottostante, mentre cercano di risolvere l'intento dell'utente di alto livello descritto con NEMO. Le modalità di implementazione e configurazione dell'infrastruttura sottostante per costruire la rete virtuale ed eseguire le operazioni sono risolte dai renderizzatori stessi. Di conseguenza, il componente Renderer fornisce un'interfaccia unificata per nascondere efficacemente la complessità sottostante. In particolare, il renderer OpenFlow ha il compito di implementare l'intento dell'utente sulla base dei dati della rete OpenFlow, che vengono ottenuti dalla risoluzione dell'intento attraverso lo User Manager, li converte in voci di flusso OpenFlow e infine li configura sugli switch OpenFlow. Il renderer si iscrive anche al Physical Network Manager per comunicare le modifiche alla rete sottostante e aggiornare, di conseguenza, le tabelle dei flussi.

- Il *Physical Network Manager* è responsabile di raccogliere e mantenere i dati, della rete fisica sottostante, nell'MD-SAL data store e di fornire questi dati ai componenti User Manager e Renderers. Poiché il gestore della rete fisica dipende dallo User Manager, che è implementato in modo indipendente dall'infrastruttura, conserva solo le parti comuni dei dati sui diversi tipi di reti fisiche, mentre le parti specifiche sono mantenute dai renderer corrispondenti. Tutti gli altri componenti che hanno bisogno dei dati dell'infrastruttura possono registrare degli ascoltatori per ricevere notifiche dal gestore delle reti.

Dall'analisi delle componenti del framework si evince un'implementazione della rete sia virtuale che fisica; infatti, per sfruttare al massimo la flessibilità del linguaggio IB-NEMO, OpenDaylight ha organizzato la rete in due livelli, quello virtuale e quello fisico.
Il livello virtuale è il primo ad essere instaziato e viene generato in risposta all'inserimento di una serie di intenti da parte dell'utente; di fatto, un utente attraverso un intento può definire nodi, connessioni e flussi virtuali con una complessità incrementale. Il livello fisico, invece, coincide con le informazioni riguardanti la topologia di rete reale messa a disposizione del sistema. Dopo aver creato la topologia virtuale NEMO esegue un mapping, tra la topologia appena creata ed il livello fisico sottostante, per realizzare una configurazione simile all'esempio illustrato in Figura 4.8.

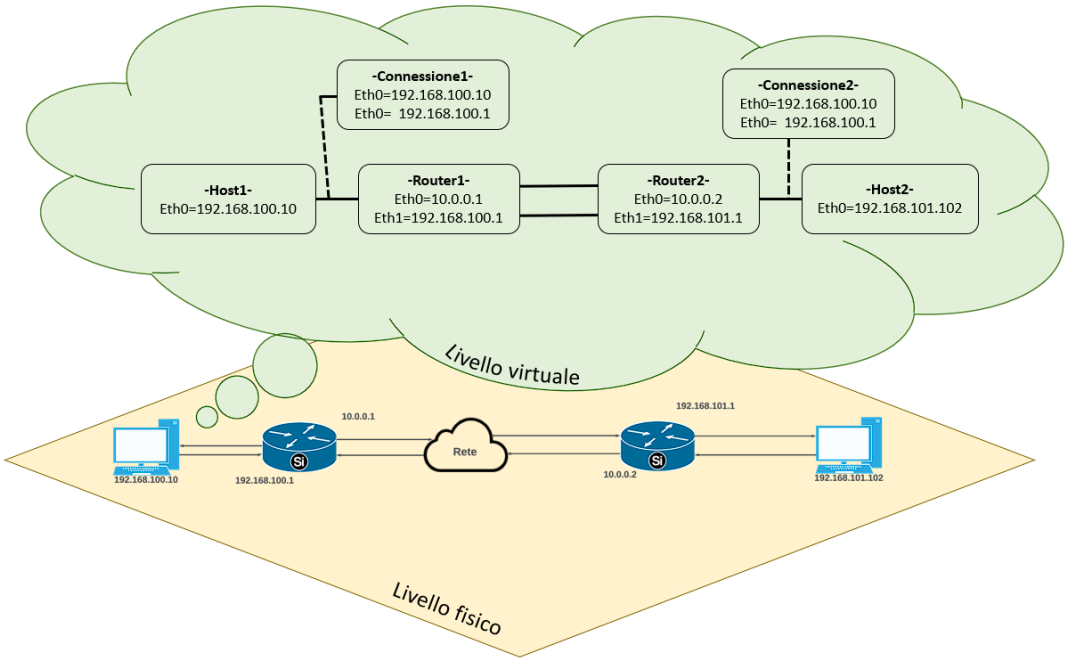


Figura 4.8: Topologia creata in OpenDayligh-NEMO[26]

Prima di concludere, risulta necessario effettuare un approfondimento sulla figura che un utente ricopre all'interno dell'architettura di OpenDayligh-NEMO.
OpenDayligh-NEMO, per semplificare la gestione dei dati, modella le informazioni accumulate dal sistemma attorno alla figura dell'utente, ovvero il sistemma permette di raccogliere i dati della rete e di associarli ad un singolo utente in modo da facilitare la creazione di più topologie virtuali sulla stessa topologia fisica.
In generale la figura dell'utente risulta essere molto complessa, di fatto un utente possiede un proprio spazio operativo, definito *User Space*, dove vengono memorizzate tutte le informazioni relative alle reti, a cui esso partecipa, ed il ruolo che ricopre all'interno di ogni rete.
I ruoli che un utente può assumere sono due, ovvero Amministratore o Utilizzatore, ma il sistema consente una definizione flessibile dei ruoli, infatti oltre a quelli di default se ne possono istaziare di nuovi, assegnando ad ognuno di essi differenti caratteristiche e funzionalità. Il ruolo aggiunge un'estrema flessibilità al sistema, infatti, con opportune configurazioni, possiamo far operare più utenti sulla stessa topologia di rete virtuale a differenti livelli di dettaglio, ad esempio permettendo modifiche, su uno spazio di indirizzi ben definito, di nodi, connessioni e policy per alcuni ruoli e non per altri.

[26]L'immagine usa le informazioni presenti in [37]

Capitolo 5

Web application Firewall

In questo capitolo andremo ad introdurre la tecnologia dei Firewall, in particolare ci soffermeremo su una sua specializzazione, il Web Application Firewall, analizzando le sue caratteristiche, la sua importanza strategica e le principali soluzioni offerte dal mercato.

5.1 Panoramica sul mondo dei Firewall

Nell'ambito delle reti di telecomunicazione, i Firewall (Figura 5.1) sono componenti software o hardware che applicano politiche di sicurezza per limitare l'accesso non autorizzato alla rete di cui fanno parte. Le politiche di sicurezza agiscono filtrando il traffico in base alle informazioni contenute in uno o più livelli dell'Open Systems Interconnection (OSI)[40], in particolare si analizzano i dati che passano attraverso i livelli applicazione, trasporto, rete e data-link. L'ipotesi di partenza è che un Fi-

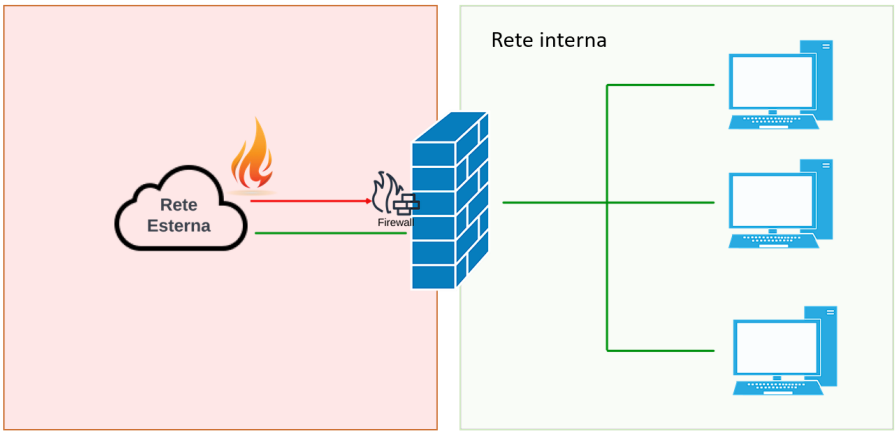


Figura 5.1: Rappresentazione cocettuale di un Firewall

rewall debba essere frapposto tra due reti, ovvero tra una rete ritenuta poco sicura, o non fidata, e una rete che riteniamo sicura, o che vogliamo proteggere. Scegliere il posizionamento del Firewall all'interno della rete risulta importantissimo, infatti si può pensare di posizionarlo come barriera dall'esterno (Internet) verso l'interno

(LAN[1]), oppure si può pensare di posizionarlo in zone diverse della stessa rete interna per creare una segmentazione in reti con differenti requisiti di sicurezza.
Le reti che vengono separate da un Firewall prendono il nome di *rete interna (o trusted) e rete esterna (o untrusted)*, dove la rete interna ha una sicurezza più elevata rispetto a quella esterna.

Alcune persone definiscono i Firewall come una scatola nera, capace di filtrare tutto il traffico Internet sia in ingresso che in uscita da una rete, la quale può essere sia comprata che costruita [41]. In realtà un Firewall è molto più complesso di una semplice scatola, di fatto può identificarsi in un sistema formato da uno o più oggetti reperibili sul mercato. Di conseguenza un Firewall va progettato scrupolosamente cercando un compromesso ottimale tra sicurezza e funzionalità, mantenendo al minimo i costi di implementazione.
Quando si progetta un firewall si devono tener bene a mente tre principi base [41]:

- il Firewall deve essere l'unico punto di contatto della rete interna con quella esterna;
- solo il traffico "autorizzato" può attraversare il Firewall. Per traffico autorizzato si intendono tutte le informazioni che rispettano le politiche di autorizzazione del traffico in un Firewall; queste possono essere suddivise in due paradigmi fondamentali:
  - whitelist: tutto ciò che non è espressamente permesso è vietato, questo approccio comporta maggior sicurezza, anche se più difficile da gestire;
  - blacklist: tutto ciò che non è espressamente vietato è permesso, questo approccio è meno sicuro, ma più facile da gestire;
- il Firewall deve essere un *sistema altamente sicuro esso stesso*. Questa affermazione si riferisce al sottosistema su cui il Firewall viene istallato, infatti molto spesso si utilizzano prodotti general purpose su cui vengono istanziati non solo i Firewall, ma anche database, server web ed altro, minando l'integrità dell'apparato e rendendo vulnerabile la rete.

Naturalmente esistono diverse tecnologie ed architetture per progettare ed implementare un Firewall, tutto dipende dal grado di sicurezza ed efficienza che il sistema deve ottenere, in particolare la tecnologia varia in base al livello dello standard OSI su cui si opera e all'architettura che si adotta.

In letteratura le tipologie di Firewall più comuni sono tre [41][42][43], ovvero filtraggio di pacchetti, Gateway a livello applicazione e Gateway di circuito.

5.1.1 Firewall a filtraggio di pacchetti

Storicamente i Router sono stati considerati i primi Firewall di rete[40], in quanto venivano utilizzati per isolare il dominio di trasmissione, limitando la comunicazione intra dominio o tra reti locali (LAN). Di fatto, la tipologia di *Firewall a filtraggio*

[1]Acronimo che identifica una Local Area Network

di pacchetti è, in senso stretto, tipicamente implementata nei Router.
Il Router a filtraggio di pacchetti, o Screening Router, osserva l'header del pacchetto IP[44] (Figura 5.2), corrispondente al Layer tre[2] del modello ISO/OSI, dove controlla quattro campi: *indirizzo di origine del pacchetto, indirizzo di destinazione, protocollo utilizzato (TCP[45], UDP[46], ICMP[47], ISAKMP[48], ecc) ed eventuali opzioni*[3].
Le decisioni di routing vengono prese valutando gli indirizzi di provenienza, in particolare se questi risultino attendibili per quella particolare destinazione, con quel particolare protocollo e opzioni. Nel caso di Firewall a filtraggio di pacchetti

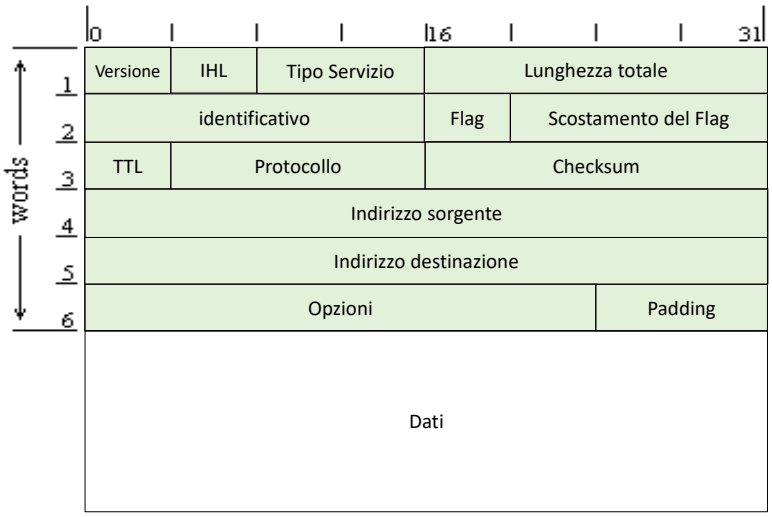


Figura 5.2: Header del pacchetto IP.

più evoluti il controllo del pacchetto avviene anche al livello quattro[4] dello standard ISO/OSI, questo implica che, oltre alle informazioni contenute nell'header IP, vengono prese in analisi anche le porte di origine e destinazione del protocollo di trasporto adottato. In questo modo, è possibile filtrare il traffico anche in base al protocollo utilizzato (HTTP[5], HTTPS[6], SMTP[7], ecc), in quanto molto spesso le applicazioni server aprono porte ben precise.
I Firewall a filtraggio di pacchetti di livello Trasporto possono essere implementati con un sistema di analisi "dinamica" o "statefull", "state-aware", ovvero capace di osservare le sessioni di comunicazione, discriminando le nuove connessioni da quelle già esistenti che vengono registrate in un'apposita tabella. Questa tecnica permette

[2]Il livello tre dello standard ISO/OSI corrisponde con il livello denominato Rete.

[3]Il campo opzioni è solitamente vuoto, ma in passato veniva utilizzato per settare il flag di source routing, che consentiva al mittente di "scavalcare" le routing tables del destinatario.

[4]Il livello quattro dello standard ISO/OSI corrisponde con il livello denominato Trasporto.

[5]https://www.rfc-editor.org/rfc/rfc9114

[6]https://www.rfc-editor.org/rfc/rfc2818

[7]https://www.rfc-editor.org/rfc/rfc5321.html

di migliorare il numero di pacchetti che si riesce ad analizzare, infatti le connessioni ritenute sicure non vengono più ispezionate approfonditamente dal Firewall, poiché già valutate all'inizio della comunicazione e confrontate con lo stato registrato. Naturalmente, vengono commercializzati anche Firewall a filtraggio di pacchetti di tipo stateless, i quali analizzano e valutano indiscriminatamente tutti i pacchetti in ingresso ad una data rete senza operare nessun tipo di ottimizzazione.

5.1.2 Gateway a livello applicazione

Un'altra tipologia di Firewall è quello definito Gateway a livello applicazione, implementato come Server Proxy[8], si comporta come una sorta di ripetitore di comunicazione. Sostanzialmente questo Firewall agisce da ponte tra l'utente e l'applicazione che vuole contattare, cioè, quando un utente apre una comunicazione viene intercettato dal Gateway che controlla il payload[9] della richiesta sia da che verso l'applicazione di destinazione, in questo modo spezza la connessione scartanto tutte le richieste sospette o malevole. Con richieste sospette o malevole si intendono tutte quelle richeste che mirano a sfruttare le vulnerabilità del software applicativo, ad esempio tramite la tecnica di buffer overflow[10].
Oltre al controllo delle richieste, un Gateway a livello applicazione è in grado di gestire gli accessi anche tramite autenticazione e non solo in base agli indirizzi IP di sorgente e destinazione. Le applicazioni che vengono protette dal Gateway devono supportare, tramite implementazioni aggiuntive, la presenza del proxy in quanto non è esattamente trasparente all'applicazione.
In conclusione i controlli messi in campo da questa tipologia di Firewall risultano essere più stringenti, di conseguenza molto più sicuri rispetto al filtraggio di pacchetti, dato che possono esaminare i dati fino al livello applicativo. Visto che client e server non parlano più direttamente, è vero che le applicazioni risultino più protette, ma un malintenzionato può comunque attaccare il proxy, nel caso si usi un canale criptato come TLS[11] o una Virtual Private Network (VPN[12]) il gateway non può fare particolari controlli e di conseguenza si comporterà come un semplice Firewall a filtraggio di pacchetti.

5.1.3 Gateway di circuito

L'ultima tipologia più comune di Firewall è il Gatway di circuito, questo si posiziona ad un livello intermedio tra il filtraggio di pacchetti e il Gateway a livello applicazione; in pratica il Gateway di circuito è visto come un proxy che svolge il

[8] https://nordvpn.com/it/blog/server-proxy/

[9] In una trasmissione informatica si indica con payload la parte di dati trasmessi destinata all'utilizzatore, a differenza degli altri elementi del messaggio necessari solo al protocollo di comunicazione.

[10] https://owasp.org/www-community/attacks/Buffer_overflow_attack

[11] https://www.rfc-editor.org/rfc/rfc8446

[12] https://aws.amazon.com/it/what-is/vpn/

lavoro del Firewall a filtraggio di pacchetti, ma applicato ai datagrammi[13], o segmenti. Per analizzare un datagramma a livello trasporto si devono accumulare più pacchetti IP per poi ricomporli così da analizzarne le informazioni. Questa tipologia di Firewall ci protegge da azioni malevole usate a livello tre, isola i server da attacchi all'handshake TCP e dalla frammentazione dei pacchetti.

5.2 Web Application Firewall

Un Web Application Firewall (WAF) è un tipo particolare di Firewall implementato come modulo di sicurezza, solitamente su un dispositivo proxy di livello applicazione, che protegge da svariati attacchi i server su cui girano applicazioni web (Figura 5.3). Il WAF è un livello di sicurezza aggiuntivo molto importante nell'ambito delle reti, perché permette di proteggersi da una serie di minacce a livello applicativo che di solito vengono ignorate dai tipici sistemi di rilevamento delle intrusioni (IDS[14]) [49], i quali si concentrano molto sull'analisi del traffico IP.
Il WAF garantisce che la sicurezza del server Web, che sta proteggendo, non sia

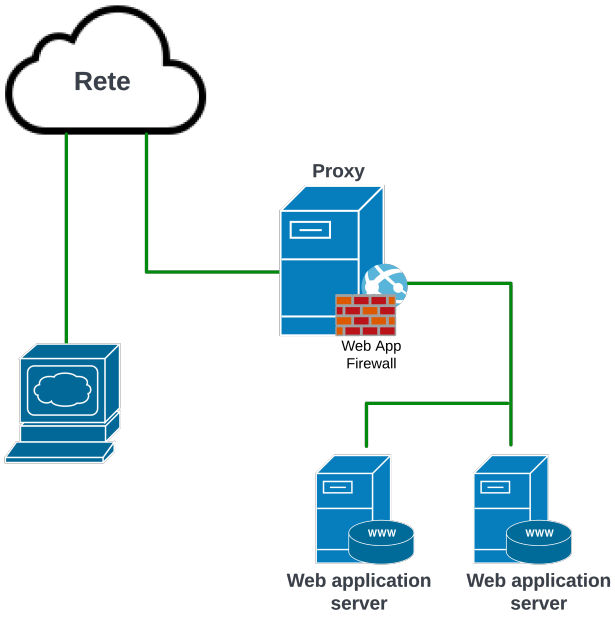


Figura 5.3: Rappresentazione cocettuale di un Web Application Firewall

compromessa esaminando i pacchetti di richieste HTTP/HTTPS, tramite Deep

[13]Il payload di messaggio inviato tramite il protocollo IP, se troppo grande, può essere suddiviso in più pacchetti detti datagrammi o segmenti, i quali vengono adeguatamente contraddistinti e numerati prima di essere inoltrati.

[14]Intrusion Detection System è un sistema che permette il monitoraggio del traffico di rete per prevenire operazioni dolose.

packet inspection[15], e i modelli di traffico Web. Quando individua qualsiasi tipo di minaccia alla sicurezza, in base al file di configurazione o al sistema di rilevamento delle intrusioni, il WAF impedisce l'attacco bloccando la richiesta HTTP, la sessione utente o il relativo indirizzo IP.
Oltre alla protezione dei server web, il WAF offre la possibilità di eseguire operazioni di controllo, in particolare permette di redarre Log[16] in tempo reale che possono essere utilizzati dai sistemi di rilevamento delle intrusioni sia per bloccare eventuali attacchi, sia per studiare comportamenti non noti che portano alla scoperta di nuove tipologie di attacchi e vulnerabilità.
Quindi, l'obiettivo principale di un WAF è la protezione delle applicazioni web già esistenti, spesso in pieno regime di funzionamento, che posseggono vulnerabilità difficili da risolvere nel breve periodo, impossibili da risolvere o risolvibili solo tramite una quantità sproporzionata di lavoro [50].
Un WAF può utilizzare due approcci molto performanti per implementare la sicurezza, infatti in aggiunta alla protezione di base detta blacklist, che si fonda sulla descrizione di schemi di attacco noti per bloccare comportamenti dolosi sul nascere, troviamo l'opzione di whitelisting.
Il whitelisting, invece di tenere conto di tutte le tipologie di attacco che esistono, si basa sulle peculiarità delle applicazioni che deve proteggere, cioè sugli input ritenuti sicuri, implementando un set di regole personalizzate che permettono di non dover aggiornare le difese per ogni nuovo attacco noto. La configurazione di whitelist offre, quindi, un'estrema flessibilità consentendo di non dover risolvere ogni minima vulnerabilità, soprattutto le cosiddette vulnerabilità Zero-Day[17]. Solitamente quando si progetta un applicazione si può decidere di implementare un WAF su cui scaricare determinati compiti, che altrimenti avrebbero gravato sulla progettazione e l'implementazione dell'applicazione stessa. Il WAF diventa quindi un punto di servizio centrale per il completamento di attività onerose per l'applicazione, ma che possono e devono essere affrontate secondo pratiche standard. Degli esempi possono essere la gestione sicura delle sessioni, tramite archivi di cookie[18], ancora l'autenticazione e l'autorizzazione centralizzate, la raccolta di tutti i messaggi di errore e dei file di log pertinenti e, per finire, l'uso di meccanismi di sicurezza pro-attivi come la crittografia degli URL.

5.2.1 Vulnerabilità e strategie di difesa

In questa Sottosezione si mette in relazione la capacità di un WAF di fornire protezione rispetto alle tipologie di attacco note più rilevanti; si specifica che non tutti i

[15]Tecnica di analisi del flusso dei pacchetti che passano nelle apparecchiature di rete, oltre l'intestazione si analizza il contenuto dei dati.

[16]Termine comunemente usato nell'informatica, derivante dal gergo nautico, che consiste nella registrazione delle operazioni che avvengono in un sistema.

[17]È una qualsiasi vulnerabilità di sicurezza informatica non espressamente nota allo sviluppatore o alla casa che ha prodotto un determinato sistema informatico. Di conseguenza non esistono contromisure note per risolvere la vulnerabilità posta in essere.

[18]I cookie sono file contenenti informazioni, che i siti web memorizzano sul computer dell'utente durante la navigazione.

WAF, disponibili sul mercato, offrono tutte le funzionalità che verranno descritte.

I cookies[19] rappresentano delle minacce concrete all'operatività di un'applicazione web, in quanto la maggior parte di queste applicazioni li utilizza come unici identificatori delle sessioni degli utenti, oltre ad utilizzarli per la raccolta di informazioni. Per queste ragioni, se un cookie viene intercettato e riutilizzato, un aggressore potrebbe essere in grado di impersonare un utente, ottenere un accesso non autorizzato ed effettuare operazioni potenzialmente catastrofiche.
In generale, un WAF può fornire diversi tipi di supporto, a protezione di queste vulnerabilità, come proteggere i cookies tramite firma e cifratura, nasconderli o rimpiazzati al bisogno e, per finire, possono essere collegati tramite una corrispondenza biunivoca al client, attraverso l'IP.

Un'altra vulnerabilità spesso presente all'interno delle applicazioni web è la così detta information leakege[20]. Questa vulnerabilità consiste nel fornire, ad un potenziale attaccante, informazioni relative alla costruzione dell'applicazione, dell'ambiente operativo o dati specifici degli utenti che utilizzano il servizio. Le informazioni che l'attaccante è in grado di reperire solitamente sono comunicazioni in chiaro riguardanti errori, informazioni di debug, commenti allegati al codice, dati sensibili degli utenti oppure è in grado di intercettare configurazioni lato server mal eseguite. Tutte queste informazioni possono essere utilizzate per penetrare all'interno del sistema che ospita l'applicazione web, così da poter effettuare diverse tipologie di attacco.
Per proteggersi da questa vulnerabilità un WAF può utilizzare il Filtro di Cloaking, ovvero quando riceve una richiesta da un qualsieasi utente, prima di inviare una risposta, maschera tutte le interazioni con il server ed i codici di ritorno HTTP, in modo da non far trapelare nessuna informazione all'esterno.

Il Cross-Site Request Forgery (CSRF[21]) è un attacco che costringe un utente finale ad eseguire azioni indesiderate su un'applicazione Web in cui è attualmente autenticato. Grazie alle tecniche dell'ingegneria sociale, come l'invio di un link via e-mail o chat, un aggressore può ingannare un utente che utilizza un'applicazione web per fargli eseguire azioni da lui selezionate. Se la vittima è un utente normale, un attacco CSRF può costringere l'utente ad eseguire richieste di modifiche di stato, come il trasferimento di fondi, la modifica dell'indirizzo e-mail e così via. Qualora un utente risulti amministratore l'attaccante può modificare lo stato dell'intero sistema.
Questa tipologia di attacco può essere impedita dal WAF tramite l'utilizzo del protocollo HTTPS, obbligatorio nelle comunicazioni sicure moderne, oppure tramite l'utilizzo di un token di sessione, ovvero vengono utilizzati dei cookies cifrati dove all'interno sono presenti dei token che devono essere inviati ad ogni interazione, per garantire il corretto funzionamento delle operazioni.

[19] https://owasp.org/www-community/controls/SecureCookieAttribute

[20] https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/05-Review_Webpage_Content_for_Information_Leakage

[21] https://owasp.org/www-community/attacks/csrf

All'interno delle applicazioni Web una vulnerabilità particolarmente critica risulta essere il caricamento dei file da parte dell'utente verso l'applicazione stessa. Spesso le applicazioni Web, per fornire particolari servizi, consentono agli utenti di caricare dei file, con diverse estensioni, per effettuare delle operazioni più o meno complesse. Un utente malevolo potrebbe inserire, all'interno del file che viene caricato, un virus per prendere il controllo dell'ambiente operativo su cui è istallata l'applicazione. Per eliminare tale vulnerabilità un WAF può utilizzare, attraverso l'Internet Content Adaptation Protocol (ICAP[22]), l'integrazione di un sistema per la scansionare dei file, in modo da validarli prima di essere elaborati.

Un altro attacco spesso utilizzato da utenti malintenzionati è il Web Parameter Tampering[23], il quale si basa sulla manipolazione dei parametri scambiati tra client e server, al fine di modificare i dati dell'applicazione, come le credenziali, permessi dell'utente, prezzo o quantità di prodotti e molto altro. Di solito, le applicazioni conservano queste informazioni all'interno dei cookies, all'interno di campi nascosti o di stringhe nelle query URL, per aumentare la funzionalità e il controllo dell'applicazione stessa.
Questo attacco può essere eseguito da un utente malintenzionato che desidera sfruttare l'applicazione a proprio vantaggio o per attaccare una terza persona. Il successo dell'attacco è strettamente collegato agli errori presenti nella logica di validazione del servizio, il quale se correttamente implementato non dovrebbe essere vulnerabile. Tuttavia, molte applicazioni di lunga data o troppo giovani, ancora in fase di testing, potrebbero riscontrare molte vulnerabilità le quali possono essere mitigate dall'utilizzo del WAF. Il WAF, infatti, oltre a convalidare i dati secondo gli input che l'applicazione dovrebbe ricevere, può utilizzare tecniche pro-attive come la crittografia dei parametri delle richieste GET e POST.
Quando un WAF effettua operazioni di validazione dei dati, seppur consente di bloccare molte operazioni dannose, è soggetto ad errori definiti falsi positivi. I falsi positivi sono operazioni lecite classificate dannose e di conseguenza bloccate, ragion per cui risultano particolarmente critiche all'interno di applicazioni business. La qualità delle operazioni di validazione dipende molto dalle caratteristiche del WAF che si utilizza e da come quet'ultimo viene configurato. Si evidenzia che nonostante il WAF sia in grado di validare i dati in ingresso all'applicazione, proteggendola di fatto dai principali attacchi di code injection, risulti necessario trovare il giusto bilanciamento tra sicurezza e affidabilità.

L'ultimo attacco di cui si parlerà in questa sottosezione è quello definito Denial of Service (DoS[24]), il quale si concentra sul rendere una risorsa, ad esempio un'applicazione web o un server, non disponibile per lo scopo per cui è stata progettata. Se un servizio riceve un numero molto elevato di richieste, potrebbe non essere più disponibile per gli utenti. Allo stesso modo, un servizio può arrestarsi se viene sfruttata una qualche vulnerabilità di programmazione o il modo in cui vengono gestite le risorse che utilizza. Per questa particolare tipologia di attacchi un WAF

[22] https://docs.vmware.com/en/VMware-NSX-Advanced-Load-Balancer/21.1.4/Configuration_Guide/GUID-21BD32AD-53A0-4661-95B0-48BCD6B1235D.html

[23] https://owasp.org/www-community/attacks/Web_Parameter_Tampering

[24] https://owasp.org/www-community/attacks/Denial_of_Service

può fare ben poco, infatti, può adottare politiche di tracciamento delle richieste e blocco per transazioni, IP o account utente che effettuano un numero troppo elevato di operazioni. Solitamente si utilizzano strumenti che monitorano l'ammontare del traffico in ingresso per gestire le connessioni e, di conseguenza, questa tipologia di attacco.

5.2.2 Tecnologie ed architetture

Fin ora sono state analizzate, in generale, le caratteristiche di un Web Application Firewall senza porre attenzione sulle tecnologie offerte dal mercato, pertanto in questa sottosezione verranno brevemente analizzate le principali architetture e soluzioni.
Solitamente un WAF può essere implementato in tre modi differenti, ognuno dei quali con i suoi vantaggi e svantaggi[25]:

- un *Network-Based WAF*, generalmente implementato su hardware dedicato, ha il vantaggio di ridurre al minimo il problema della latenza essendo installato localmente, tuttavia rappresenta l'opzione più costosa che richiede un certo spazio fisico sia per essere installato, che per effettuarne la manutenzione;
- un *Host-Based WAF* può essere completamente integrato nel software di un'applicazione e non necessita di hardware dedicato. Questa soluzione è meno costosa di un Network-Based WAF e offre una maggiore personalizzazione, di contro possiede alcuni svantaggi, come il consumo di risorse del server locale, la complessità dell'implementazione, i costi di manutenzione e, in linea di massima, tempi di progettazione molto lunghi;
- un *Cloud-based WAF* è un alternativa decentralizzata e meno costosa rispetto alle precedenti, offre un'opzione conveniente in termini di costi e risulta facile da implementare. Questa soluzione ha il vantaggio di essere continuamente aggiornata dal fornitore del servizio, può essere configurata velocemente, può prevedere configurazioni personalizzate e un costo minimo iniziale. Naturalmente anche questa tecnologia non è esente da svantaggi, ad esempio, oltre al problema dell'elevata latenza, gli utenti affidano la responsabilità di proteggere la propria applicazione ad un ente terzo, per cui alcune funzionalità del WAF potrebbero non essere note.

Valutate le tecnologie di implementazione di un WAF, si possono annoverare, brevemente, le principali architetture con cui viene istanziato [51]:

- *Inline*: in questa modalità un dispositivo WAF si colloca nel mezzo del traffico tra un utente e un'applicazione web, consentendo di ispezionare e bloccare gli attacchi in modo trasparente ai server web. Questa tecnica, a seconda di come è implementata, può generare gravi problemi di latenza;

[25] https://www.cloudflare.com/it-it/learning/ddos/glossary/web-application-firewall-waf/

- *Out of Band*: in questa modalità il WAF non si frappone tra i due interlocutori, bensì riceve una copia del traffico in modo da annullare il problema della latenza. Questa soluzione implementativa non consente un blocco preventivo di comportamenti dolosi, in quanto il WAF, che sia hardware o software, reaggirà sempre in ritardo rispetto l'attacco;

- *Agent*: quando si utilizza la modalità Agent, il WAF è inserito come modulo software nel server web, o in un reverse proxy[26], imitando una modalità Inline con una configurazione hardware dedicata. Sebbene questo consenta un posizionamento di rete più semplice, può diventare molto invasivo in termini di risorse generando un problema soprattutto nella gestione delle prestazioni.

- *Cloud*: questa modalità coincide con la tecnologia Cloud-based WAF già analizzata.

Le architetture e tecnologie analizzate portano con sè innumerevoli vantaggi e svantaggi, ma in generale l'adozione di un WAF comporta rischi e complicazioni aggiuntivi che non devono essere sottovalutati.
Secondo l'Open Web Application Security Project (OWASP) [50] le complicazioni iniziano già a livello di progettazione, infatti nasce la necessita di capire che tipologia di WAF adottare, che configurazioni dovranno essere implementate, come dovrà essere addestrato il WAF e per finire come dovrà essere testato. Di fatto un WAF rende molto più difficile effettuare le fasi di test per un dato servizio web, in quanco configurazioni sbagliate possono portare il blocco delle sessioni di più utenti o addirittura il blocco dell'intero canale di comunicazione. La difficoltà consiste nell'identificare chiaramente i tentativi di intrusione verso il sistema, giacché le così dette "minacce fasulle" rappresentano un vero e proprio inconveniente sia per i progettisti che per gli sviluppatori, dal momento che bloccano comunicazioni innocue scambiandole per reali minacce.

In conclusione quando si vuole implementare un WAF bisogna valutare in fase di progetto pro e contro, ragionando anche in base al budget che si possiede per lo sviluppo.

5.3 Implementazioni

In questa sezione ci concentreremo sulle implementazioni software del Web Application Firewall, in modo da avere una visione chiara delle opzioni che un individuo possiede per adottare questa tecnologia. Si evidenzia che saranno trascurate le implementazioni hardware, in quanto non utili allo sviluppo di sistemi basati su intenti, poco flessibili dal momento che sono legati a venditori specifici e non pertinenti alle scelte adottate in questa tesi.

Molte organizzazioni hanno sviluppato il proprio WAF, differenziandosi per capacità di protezione, latenza ed efficienza. Oltre alle principali grandi aziende

[26] https://www.nginx.com/resources/glossary/reverse-proxy-server/

Tecnologiche, come Amazon[27], Microsoft[28], Cisco[29] eccetera, che forniscono questo servizio a pagamento e naturalmente senza divulgare la propria implmenetazione, risultano particolarmente rilevanti le organizzazioni che sviluppano i così detti WAF Open-Source [52].
I WAF Open-Source sono dei software che vengono continuamente migliorati grazie allo sforzo congiunto di organizzazioni private, che hanno interesse a migliorare il software per impieghi commerciali, e community di programmatori indipendenti. Questo approccio permette, in senso stretto, di avere enormi vantaggi sia in termini prestazionali che in termini di protezione, quali [52]:

- *trasparenza*: il codice sorgente è accessibile liberamente, chiunque può esaminare le modifiche approtate al codice, la sua storia e i suoi progressi;
- *affidabilità*: a differenza del codice proprietario, vincolato ad un'unica azienda che ha il compito di aggiornarlo e manutenerlo, un codice open source fa affidamento sul contributo della community che lo supporta, aggiornandolo continuamente. Gli standard aperti e la peer review intrinseca nell'open-source garantiscono che il codice venga revisionato in modo appropriato e con regolare frequenza risolvendo tutte le problematiche;
- *convenienza economica*: un'azienda può risparimiare sull'acquisto del software, adottandone appunto uno open source, ed affidarsi a società di supporto per configurazioni ed eventuali problematiche, abbattendo notevolmente i costi.
- *flessibilità*: grazie alla natura dell'open source, ogni individuo o azienda può modificare il codice per adattarlo alle proprie esigenze, rilasciandolo eventualmente alla community per ottenere revisioni e correzioni di eventuali problemi.

Il mondo del software libero è vasto e i WAF Open-Source non fanno eccezione, ma in questo progetto si prenderanno in esame le tre principali soluzioni offerte dal mercato, cioè NAXSI WAF, Shadow Daemon WAF e ModSecurity WAF, il quale sarà anche utilizzato nell'implementazione della soluzione.

5.3.1 NAXSI

NAXSI [53] acronimo di Nginx Anti XSS and SQL Injection, è un WAF basato su un modulo Nginx fornito in bundle con diversi sistemi operativi UNIX-like e OPNsense[30]. Per impostazione predefinita, questo modulo legge una selezione limitata di regole di base che coprono molti dei modelli noti associati alle vulnerabilità dei

[27] https://aws.amazon.com/it/?nc2=h_lg

[28] https://azure.microsoft.com/en-us/

[29] https://www.cisco.com/

[30] https://opnsense.org/

siti web.
Il WAF può esaminare molte tipologie di dati, tra cui URL, parametri di richiesta, cookie, intestazioni e corpo di richieste POST, può essere attivato o disattivato semplicemente modificando la configurazione di Nginx. La generazione automatica di whitelist semplifica l'implementazione del firewall upstream ed elimina quanto più possibile le minacce fasulle.
Il software possiede un set di regole di base espandibili dall'utente e applicativi di supporto, come NX-Utils e Doxi che semplificano la produzione di report e regole. NAXSI possiede due tipologie di regole, rispettivamente *Main Rules*, utilizzate per bloccare i principali attacchi di injection, e *Basic Rules*, quest'ultime tipicamente utilizzate per inserire in whitelist regole primarie o supplementari. Per concludere contrariamente alla maggior parte dei Web Application Firewall, NAXSI non si basa su un sistema signature base come un normale antivirus, ragion per cui non può essere aggirato da uno schema di attacco "sconosciuto".

5.3.2 Shadow Daemon

Shadow Daemon [54] può essere inteso come una suite di strumenti progettati per identificare, registrare e prevenire gli attacchi alle applicazioni Web. Inquadrato, tecnicamente, come un WAF è capace di intercettare le richieste e rimuove i parametri ritenuti dannosi. È una soluzione modulare che isola le applicazioni e le interfacce per aumentare la sicurezza, la flessibilità e la scalabilità.
Shadow Daemon è semplice da installare e da gestire, grazie ad un'interfaccia online ben organizzata che contiene anche una panoramica approfondita sulle minacce note e come proteggersi. Tutt'oggi, questo software open source è supportato solo da alcune librerie per linguaggi di programmazione e framework, riuscendo a garantire una protezione esaustiva solo per le minacce note più importanti, quali code injection, file upload e controllo su backdoor (vulenrabilità introdotte nella Sezione 5.2.1).
La peculiarità di Shadow Daemon è che, a differenza di molti altri WAF, non blocca completamente le richieste dannose o dolose, permettendo, quando possibile, l'inoltro delle informazioni rimuovendo solo i componenti potenzialmente dannosi; questa caratteristica è il suo cavallo di battaglia, in quanto previene gli attacchi senza gravare inutilmente sugli utilizzatori del servizio in caso di minacce fasulle.

5.3.3 ModSecurity

ModSecurity, o Modsec [55], è un Web Application Firewall Open-Source originariamente progettato come modulo per il server Apache, si è poi evoluto per fornire una serie di funzionalità di filtraggio delle richieste e delle risposte del protocollo HTTP, insieme ad altre caratteristiche di sicurezza, su una serie di piattaforme diverse, tra cui Apache HTTP Server, Microsoft IIS e Nginx[31].

Le principali caratteristiche di ModSecurity riguardano il monitoraggio real-time delle applicazioni web e i controlli di accesso, ma consente anche di effettuare

[31] https://www.nginx.com/

il logging del traffico e utilizzare una tecnica chiamata Virtual patching.
Il Virtual patching è il processo di creazione di una politica temporanea utilizzata per mitigare i rischi associati alla scoperta di nuove vulnerabilità della sicurezza in una applicazione Web esistente; sostanzialmente si esegue il monitoraggio del traffico cercando di fermare specifiche richieste dolose in attesa di un aggiornamento ufficiale del software.
Modsecurity nel monitoraggio real-time e nelle tecniche di Virtual patching eccelle, questo perchè utilizza delle configurazioni flessibili e delle regole di funzionamento, che possono essere facilmente definite dal singolo utente. Le configurazioni modellano come il software debba elaborare i dati che riceve, invece le regole sono utilizzate per decidere cosa fare con i dati elaborati.
La piattaforma fornisce un linguaggio di configurazione delle regole noto come "SecRules" e per analizzarlo si può partire dalla composizione generale di una regola [55]:

```
1 SecRule VARIABLES OPERATOR [ACTIONS];
```

Osservando la composizione generale si nota subito la dicitura iniziale "SecRule" anteposta prima di ogni nuova regola per scandirne l'inizio, subito dopo troviamo l'inserimento di una *VARIABLES*, che indica quale parte della richiesta deve essere elaborata, seguita da un *OPERATOR*, ovvero un'espressione regolare, pattern o parola chiave da controllare nelle variables, e un *ACTION*, opzionale, che specifica come la richiesta debba essere gestita in caso di riscontro positivo tra contenuto della richiesta HTTP ed operator.
Di seguito viene riportato un esempio utilizzato per comprendere, in toto, come costruire una regola e come questa venga applicata all'interno del sistema:

```
1 SecRule ARGS:/jform\[username\]/ "[^a-zA-Z]"
    "t:none,phase:2,deny,id:6,log,msg:'Errore, l'id si compone solo di
    caratteri.'"
```

Come precedentemente specificato, la regola parte sempre con la stringa di inizio SecRule, dopo di che vengono inserite le variables e gli operator.
In questo caso, la regola analizza il payload di una richiesta, in particolare con la variables *ARGS:/jform\[username\]/* si specifica la volontà di analizzare il campo "username", presente all'interno del form sottomesso per mezzo della richiesta HTTP, tramite l'espressione regolare *"[^a-zA-Z]"*, implementata come operator. Nel momento in cui la richiesta è processata e l'operator verifica la presenza di caratteri non validi chiama le action, opzionali e specificate in coda alla regola. Nel caso in esame si esprimono una catena di azioni, in particolare l'azione "phase:2" specifica che le operazioni devono avvenire in fase di analisi del corpo del messaggio, l'azione "deny" specifica di scartare la richiesta qualora non rispetti la regola, l'azione "log" impone la scrittura del messaggio "msg" all'interno del file log di sistema ogni volta che la regola viene attivata e, per finire, troviamo un campo "id" utilizzato per identificare univocamente la regola.

Naturalmente si possono istanziare configurazioni e regole completamente personalizzate costruendole da zero, oppure si può optare per un insieme di configurazioni e regole già create dalla community di supporto. Tra le varie configurazioni messe a dispozione spicca l'OWASP ModSecurity Core Rule Set[32] (CRS), si tratta di un insieme open-source di regole scritte nel linguaggio SecRules di ModSecurity, che mira a proteggere le applicazioni Web da un'ampia gamma di attacchi, inclusa la Top Ten di OWASP[33], gestendo un minimo anche le minacce fasulle.

In conclusione ModSecurity è stato scelto come WAF da istanziare nella soluzione di questa tesi per via dei suoi grandi vantaggi, come rilasciato tramite licenza open-source, grande supporto da parte della community, adesione ai più noti standard di sicurezza e, per finire, performance molto elevate rispetto alla concorrenza.

[32] https://owasp.org/www-project-modsecurity-core-rule-set/

[33] https://owasp.org/www-project-top-ten/

Capitolo 6

Design della soluzione

In questo capitolo si discuteranno i dettagli progettuali della soluzione di questo lavoro, che ha come scopo quello di realizzare un sistema di configurazione di funzioni di sicurezza virtualizzate, basato su intenti. In particolare, lo scenario che si vuole delineare riguarda la creazione, tramite intenti, di un Web Apllication Firewall come una vNSF, istanziata da un generico utente su un provider a protezione di un servizio web.

6.1 Design di alto livello

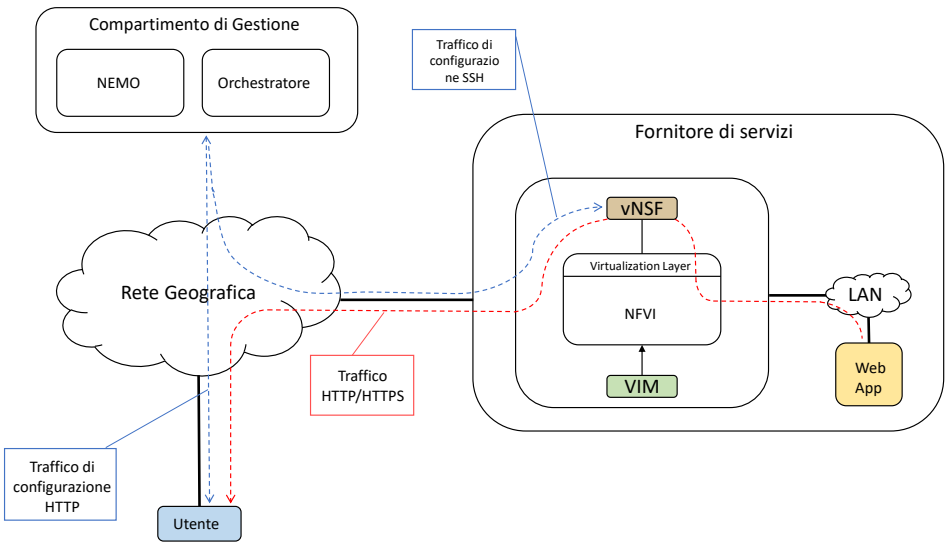


Figura 6.1: Visione del Design ad alto livello.

Lo schema in Figura 6.1 è la rappresentazione ad alto livello dell'architettura della soluzione, qui troviamo il flusso principale della comunicazione tra i servizi basati sugli intenti, che vengono utilizzati per configurare le vNSF su una piattaforma NFV, ampiamente analizzata nel Capitolo 2.

L'architettura è composta da tre attori principali, rispettivamente dall'*utente*, in grado di richiedere la generazione delle configurazioni delle vNSF e le relative instanze, dal *Sistema di gestione*, che si occupa di gestire ed istanziare tutte le richieste dell'utente, ed infine dal *Fornitore dei servizi* che mette a disposizione la piattaforma NFV dove verrà istanziata la vNSF.
L'idea di base è quella di permettere un'interazione funzionale tra Sistema di Gestione ed utente, il quale tramite richieste basate su intenti è in grado di definire la tipologia di sicurezza da applicare al suo servizio web; naturalmente l'architettura può essere facilmente estesa per integrare qualunque tipo di vNSF o VNF. Il Sistema di gestione interpreta l'intento e genera delle configurazioni abbastanza "generiche", che possono essere poi istanziate a basso livello sul Fornitore di servizio specificato dall'utente. Per comodità in questo scenrio all'interno del Sistema di Gestione vengono configurate anche le caratteristiche della struttura dell'NFV che serviranno per istanziare la vNSF.

Nella Figura 6.1 viene, inoltre, mostrato ad alto livello il traffico scambiato dalle componenti funzionali, suddiviso in quello utilizzato per le configurazioni, rappresentato in blu, e quello utilizzato per lo scambio dati, rappresentato in rosso.
In questo particolare scenario il traffico dati ha origine da un utente e, senza coinvolgere il Sistema di Gestione, prima di giungere al servizio web target attraversa le risorse di rete della NFVI. Quindi, il traffico dati viene elaborato dalla piattaforma NFV e raggiunge la vNSF, qui il traffico viene esaminto e solo alla fine dei controlli viene inoltato tramite rete locale al servizio web target. Nel caso in esame la vNSF si trova nella stessa rete del servizio web, ma nulla ci vieta di dislocare i servizi, con tutte le problematiche che ne conseguono.
Il traffico di configurazione, invece, parte dall'utente verso il Sistema di Gestione e da quest'ultimo verso il Fornitore di servizi. Risulta chiaro che l'utente è disaccoppiato dal fornitore di servizi e, in linea di massima, si limita soltanto a specificare gli obiettivi di sicurezza che vuole raggiungere senza specificare come farlo. Bisogna notare che anche il traffico di configurazione deve passare attraverso le risorse di rete della NFVI per raggiungere, istanziare e configurare la vNSF.

Nel prosieguo del capitolo si analizzeranno singolarmente tutti i moduli e sottomoduli funzionali, per comprendere appieno come l'implementazione debba essere organizzata e che requisiti rispettare.

6.2 Sistema di Gestione

Il Sistema di Gestione è rappresentato, nella Figura 6.1, come identificativo generico per un componente logico, che ha il compinto di gestire e configurare i servizi di sicurezza all'interno dell'architettura proposta. Questo elemento è composto da due moduli, ciascuno con caratteristiche e compiti ben distinti, rispettivamente il framework NEMO, ampiamente introdotto nel Capitolo 4, che permette la generazione di file di configurazione ed un modulo di orchestrazione che si occupa della gestione ed istanziazione delle configurazioni.

Il primo modulo è utilizzato per realizzare un sistema di configurazione per i servizi di sicurezza, vNSF, richiesti dall'utente. Per fare ció NEMO espone, tramite un'interfaccia Northbound intent-based, una serie di canali di comunicazione

che permettono di richiedere, con un linguaggio dichiarativo come IB-NEMO [34], un servizio di sicurezza personalizzato con delle configurazioni espresse dall'utente stesso. In sostanza, il framework NEMO ha il compito di tradurre gli intenti dell'utente in file di configurazioni generici che saranno poi passati al modulo di orchestrazione, sarà quest'ultimo ad installare e configurare la vNSF corrispondente nell'infrastruttura appropriata. Questo approccio ci consente di poter gestire la configurazione delle vNSF in modo centralizzato ed automatizzato. Naturalmente NEMO, in questo scenario, è stato modificato per implementare i WAF come servizi di sicurezza, ma come già detto è possibile estenderne le funzionalità per gestire più tipologie di vNSF.
Il secondo modulo, invece, consiste in un orchestratore incaricato di prelevare il o i file di configurazione prodotti da NEMO per generare le opportune istanze. Come già accennato, per questioni di semplicità implementative, l'orchestratore deve poter comunicare direttamente con tutti gli attori dell'architettura e, per queste ragioni, implementa anche la capacità di interaggire con la piattaforma messa a dispozione dal Fornitore di servizio.

6.2.1 Modulo NEMO

In questa Sottosezione si analizzerà approfonditamente NEMO, concentrandoci sulle modifiche necessarie per il suo funzionamento, all'interno della soluzione proposta, e sulla comparazione che verrà effettuata tra l'architettura di partenza (Sezione 4.2.2) e quella modificata.

Per selezionare il framework di gestione degli intenti si è svolta un'analisi approfondita non solo delle tipologie di linguaggio esistenti, ma anche dei linguaggi stessi cercando di individuare punti di forza e debolezza su cui far pendere l'ago della bilancia. Le peculiarità ricercate nel linguaggio sono state grande capacita auto-esplicativa e semplicità di estensione, infatti grazie a queste caratteristiche si può incrementare notevolmente l'usabilità dell'interfaccia utente. Per queste ragioni, il linguaggio selezionato per implementare il gestore degli intenti è stato proprio NEMO; questa scelta è stata fatta anche in relazione all'implementazione realizzata da OpenDaylight, la quale è inserita nei paper dell'IETF e risulta in corso di standardizzazione.

Come si può osservare, nella Figura 6.2 sono rappresentate, ad alto livello, le componenti del modulo NEMO divise fondamentalmente in due gruppi; il primo gruppo si occupa di prelevare l'intento, analizzarlo e configurarlo come oggetto complesso all'interno della base dati, in modo da istanziare la topologia di rete virtuale, introdotta nella Sezione 4.2.2. Per quanto riguarda il secondo gruppo di componenti, queste hanno il compito, previa richiesta formale tramite API REST, di analizzare la base dati ed istanziare dei file di configurazione che possono essere utilizzati per realizzare un servizio di sicurezza.
Si osserva, già in questa prima introduzione, che il modulo NEMO a differenza di OpenDaylight NEMO non sfrutta appieno la componente di Physical Network Manager, infatti questa componente ha il compito di gestire la comunicazione con gli apparati fisici, rilevando la topologia di rete sottostante per mapparla secondo le indicazioni presenti nel *VN Space*, e delegando all'opportuno Rederer l'istanza

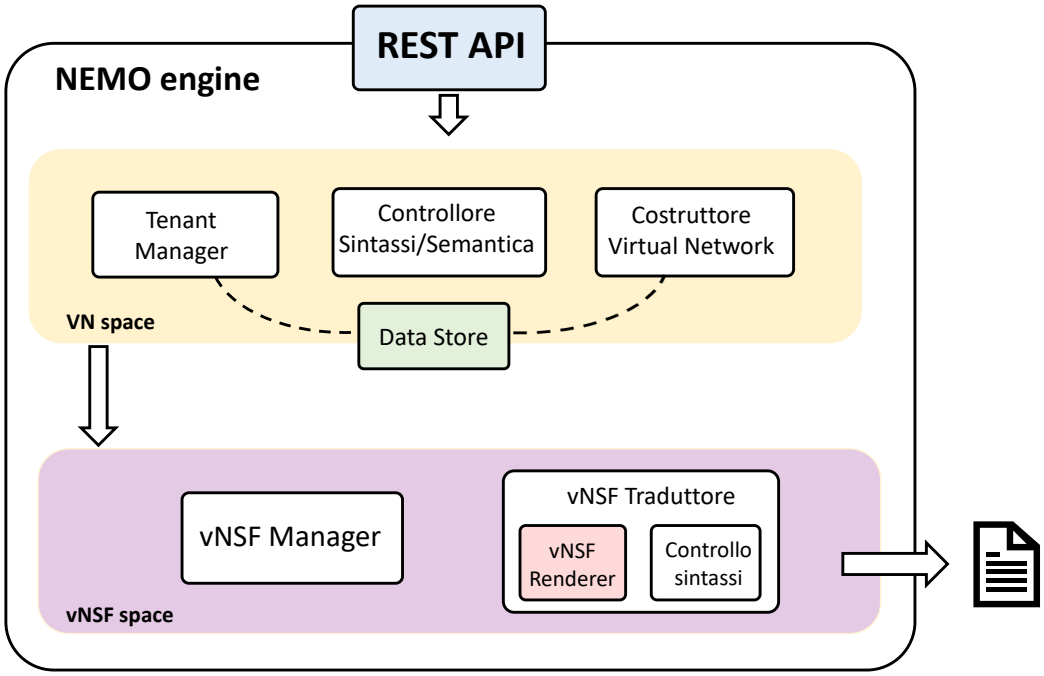


Figura 6.2: Design logico modulo NEMO[1].

dell'oggetto modellato. Nell'architettura di OpenDaylight, quindi, possiamo ritrovare due Renderer, a seconda del protocollo applicato all'interfaccia Southbound, rispettivamente *CLI Renderer*, che riceve in ingresso una serie di comandi shell[2] per effettuare le cnfigurazioni dei dispositivi, e *OpenFlow Renderer* per configurare gli switch basati su tecnologia SDN. Analizzando le funzionalità offerte da queste componenti si riscontra un'inadequatezza funzionale che si allontana dallo scopo di questo lavoro. L'approccio utilizzato nella soluziona proposta, di fatto, prevede che le problematiche sopra citate vengano risolte dall'orchestratore, sfruttando si il NEMO engine, ma delegando all'esterno la parte di gestione e implementazione delle funzioni di sicurezza.

Riprendendo l'analisi dell'architettura, procedendo dall'alto verso il basso, il primo elemento che incontriamo riguarda le API REST, queste API sono un insieme di metodi di alto livello che l'utente può utilizzare per interfacciarsi con la piattaforma e le sue strutture dati. Le API REST operano in maniera diretta sul *Tenant Manager* (o User Manager) [38], suddiviso a sua volta in tre componenti con due funzioni principali, la prima si occupa di gestire gli utenti tramite una declinazione del protocollo Autenticazione, Autorizzazione, Accounting (AAA) [56], invece la seconda si occupa di gestire le richieste fatte dall'interfaccia smistandole sull'opportuna funzione.
Il Tenant Manager, inoltre, è in grado di immagazzinare le informazioni che riceve dagli intenti all'interno del *Data Store*, mappandole sull'entità utente che le ha generate. Questo approccio garantisce una proprietà molto importante, ovvero la *multi tenancy*, che associa ogni informazione passante attraverso l'interfaccia

[1]L'immagine usa le informazioni presenti nel manuale utente di OpenDayligh [38]

[2]Comandi che possono essere utilizzati nel terminale dei sistemi linux-based.

all'utente che l'ha definita, in questo modo la piattaforma può gestire più scenari e informazioni contemporaneamente, restituendo configurazioni diverse in base all'utente che le richiede.

Il terzo modulo che incontriamo è il *Controllore di Sintassi/Semantica*, il quale fornisce supporto al Tenant Manager esaminando le richieste che superano il vaglio del protocollo AAA.
Come già detto nel Capitolo 4 il Tenant Manager è in grado di supportare due tipologie di richieste, language-style o structure-style, ragion per cui il Controllore di Sintassi/Semantica deve integrare due handler differenti per la verifica. Quindi, dopo che le richieste sono state prese in carico, prima di essere smistate sulla funzione di pertinanza, questo modulo effettua un controllo sia sintattico che semantico per correggere sul nascere eventuali errori commessi dall'utente.
Grazie all'utilizzo del parser JavaCC, NEMO verifica la correttezza di ogni singola parola ed entità presente nell'intento, rispettando i limiti imposti dal linguaggio utilizzato e dalle strutture già implementate dentro NEMO.

L'ultimo modulo che incontriamo nel VN space è il *Virtual Network Builder* che si occupa della gestione della rete virtuale.
Questo modulo, di fatto, modella la rete virtuale in base agli intenti registrati nella Base Dati del framework, dopo di che si occupa di interaggire con il Physical Network Manager per mappare le istanze virtuali generate sulla rete fisica sottostante.

I moduli analizzati fin ora fanno parte dell'architettura originale di OpenDaylight NEMO e nella soluzione progettuale sono stati completamente riutilizzati, la seconda parte dell'architettura in Figura 6.2, invece, mostra i moduli che sono stati aggiunti per plasmare il comportamento del framework in base alle esigenze elencate all'inizio di questo lavoro di tesi.

Una delle componenti fondamentali sviluppate è il *vNSF Manager*, il quale può essere contattato direttamente dall'interfaccia REST, dopo che il Tenant Manager e il Controllore di Sintassi/Semantica abbiano verificato la leggimità della richiesta. Questo modulo prende in carico le richieste espresse da parte dell'utente, riguardanti il servizio di sicurezza che vuole istanziare, raccogliendo le informazioni dalla Base Dati riferite a tutte le proprietà del servizio di sicurezza specificato, controllando che siano effettivamente lecite, e creando una struttura dati consistente che verrà poi inviata al *vNSF Translator*.
Il vNSF Translator ha il compito di prendere la struttura dati creata dal vNSF Manager ed effettuare un controllo semantico basato su un modello precedentemente inserito all'interno di NEMO; dopo la prima fase, se non sono presenti errori nelle proprietà o nelle caratteristiche del servizio di sicurezza indicato, concretizza la struttura dati, tramite l'*vNSF Renderer*, in un file di configurazione generico che dovrà essere successivamente tradotto ed adattato all'infrastruttura target. Questa è stata una scelta obbligatoria, dal momento che si voleva rendere il framework NEMO il più indipendente possibile dall'architettura delle piattaforme NFV presenti sul mercato, in linea di principio si vuole costruire una sorta di "semilavorato" che dovrà poi essere istanziato a basso livello da qualcun altro, nel nostro caso dall'orchestratore.

Per quanto riguarda la comunicazione dei moduli sopra introdotti, ognuno di loro sfrutta le caratteristiche del linguaggio *Object Oriented*, come Java, per lo scambio delle informazioni; infatti grazie all'utilizzo di metodi pubblici ogni componente è in grado di utilizzare i servizi offerti dalle altre componenti.
Il modulo Tenant Manager, oltre alla possibilità di sfruttare le peculiarità del linguaggio Object Oriented, espone un interfaccia verso le API REST, basta su remote procedure calls, che permette una comunicazione asincrona tra le parti.

6.2.2 Modulo Orchestratore

Per completare la panoramica sul Sistema di Gestione analizzeremo nel dettaglio il modulo Orchestratore, tralasciando i dettagli implementativi ed approfondendo le funzionalità di ogni suo singolo elemento.

Come ampiamente anticipato, NEMO non deve essere in grado di legarsi ad una specifica piattaforma NFV, bensì deve mantenere comportamenti generici che possono essere applicati ad una moltitudine di piattaforme; per queste ragioni è stato introdotto il modulo di orchestrazione, il quale permette l'istallazione diretta delle configurazioni di una vNSF su una specifica piattaforma NFV, nel caso di questo progetto Kubernetes.
L'architettura del modulo, visibile in Figura 6.3, è composta da due moduli logici, rispettivamente *Pipeline di gestione delle configurazioni* e *Client di configurazione*.

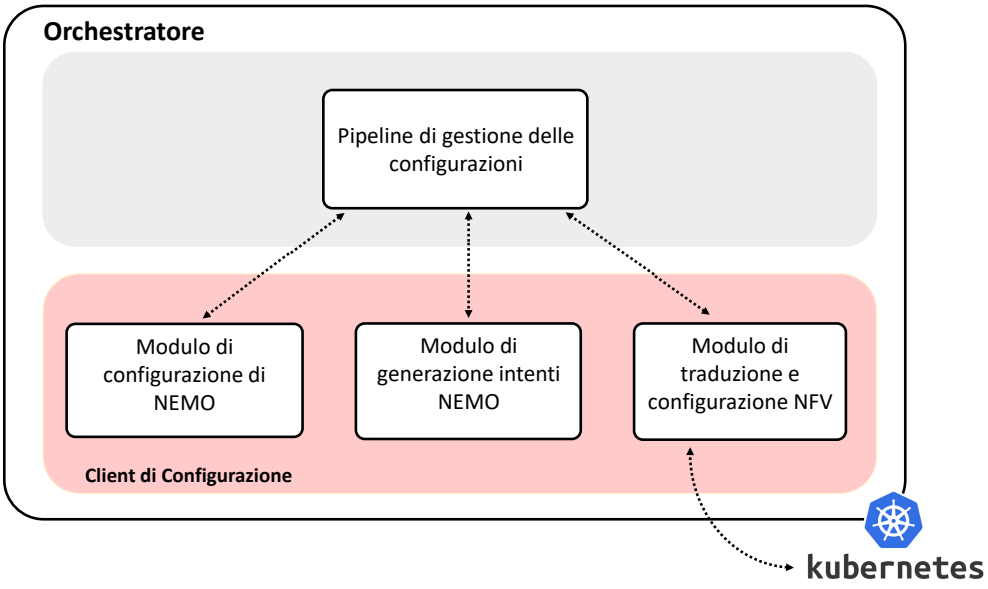


Figura 6.3: Architettura ad alto livello dell'Orchestratore.

Il modulo Client di configurazione, aiutato anche dal protocollo SSH [57], ha il compito di interfacciarsi con il framework NEMO, con la piattaforma NFV target e con la singola istanza della vNSF. Questa componente logica, per realizzare le funzionalità appena accennate, si compone di tre moduli funzionali.
In ordine, il primo modulo che viene utilizzato è il *Modulo di configurazione di*

NEMO, il quale ha il compito di fornire una configurazione preliminare delle entità, ovvero definisce le entità che saranno poi utilizzate per mappare nello spazio virtuale gli elementi della topologia desiderata. Questo modulo sfrutta un template di elementi modificabile, ed iniettabile in NEMO, per permettere una qualsiasi estenzione delle entità già presenti nel framework (per aggiungere entità ex novo fare riferimento all'Appendice B).

Il secondo modulo funzionale, definito come *Modulo di generazione intenti NEMO*, ha il compito di interfacciarsi con le API REST offerte dal framework per inviare comandi scritti in linguaggio dichiarativo. Questa componente risulta essere di fondamentale importanza, in quanto, grazie all'invio multiplo di intenti, consente un automazione dei test di interazione con l'intero sistema. Il risultato dell'interazione del secondo modulo funzionale con NEMO genera una rappresentazione virtuale della topologia specificata tramite intenti.

Il terzo ed ultimo modulo del Client di configurazione è il *Modulo di traduzione e configurazione NFV*. Questo componente funzionale si occupa di chiedere a NEMO la configurazione del servizio di sicurezza specificato dall'utente e, tramite la generazione di un canale SSH, aprire una comunicazione verso la piattaforma NFV sia per controllare lo stato dei servizi attivi sia per generare o modificare il comportamento della vNSF opportuna.

Di base i moduli del Client di configurazione non comunicano tra di loro ed operano in autonomia, per ovviare a questa problematica si introduce il modulo *Pipeline di gestione delle configurazioni* che, come anticipato dal nome, esegue i moduli sopracitati in pipeline per ottenere un ciclo di istanziazione completo del servizio di sicurezza. Anche in questo caso la componente è finalizzata allo scenario specifico di questo lavoro, ma nulla vieta di espandere le sue funzionalità.
Quindi, l'orchestratore deve avere un idea di base di come la rete e i servizi di sicurezza devono essere organizzati, ma è indipendente da esso, frapponendosi come tramite tra l'orchestratore della piattaforma NFV, che regola la rappresentazione fisica della topologia, e la topologia virtuale istanziata dentro NEMO.

6.3 Fornitore di servizi

Procedendo nell'analisi dell'architettura della soluzione analizziamo il modulo Fornitore di servizi, il quale mette a disposizione la piattaforma su cui dovranno essere istanziate le vNSF.

Nella progettazione della soluzione non sono stati presi in considerazione scenari in cui si sviluppano catene di servizi di sicurezza complessi[3], bensì si considera l'istanziazione di un servizio di sicurezza alla volta. Per queste ragioni si opera utilizzando un immagine docker del servizio di sicurezza, semplificando, in questo modo, le procedure di istallazione e messa in opera.

[3] https://www.cisco.com/c/dam/en_us/training-events/le31/le46/cln/pdf/webinar_slides/ccie_et_sfc_joe_clarke.pdf

6.3.1 Architettura NFV e progettazione della vNSF

Come ampiamente anticipato, lo scenario di implementazione della soluzione vuole avvicinarsi il più possibile ad uno scenario implementativo reale, in modo da verificare vantaggi e svantaggi che la soluzione posta in essere porta con se.

Kubernetes è stata la piattaforma NFV selezionata per questo lavoro, in quanto offre un sistema di orchestrazione che aiuta nella distribuzione, nella scalabilità e nella gestione delle applicazioni, sia web che di rete. Di fatto, nell'architettura NFV, Kubernetes svolge un ruolo fondamentale come Virtual Infrastructure Manager (VIM) e VNF Manager (VNFM).
Una delle caratteristicche più importanti di Kubernetes è fornire un ambiente comune ad un'ampia gamma di infrastrutture cloud, infatti è supportato dai principali IaaS commerciali come Microsoft Azure[4], Amazon AWS[5] e Google cloud[6]. L'ecosistema Kubernetes, in aggiunta, ci permette di distribuire e gestire rapidamente applicazioni con carichi di lavoro nativi su container, riuscendo anche ad incapsulare, nei suddetti container, macchine virtuali preesistenti utilizzando il plugin *KubeVirt*. In definitiva questa piattaforma consente di orchestrare un'ampia gamma di carichi di lavoro, come VNF, CNF, vNSF, VM, container e funzioni virtualizzate (Figura 6.4).
Questa piattaforma è stata selezionata dal momento che, essendo un codice open

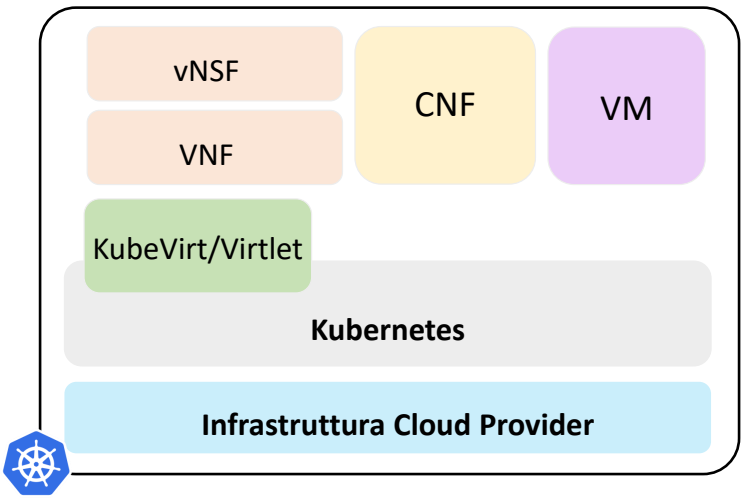


Figura 6.4: Moduli e carichi di lavoro supportati da kubernetes.

source ampiamente supportato e utilizzato nel mercato moderno, abbraccia in toto le tematiche e le esigenze di questa testi garantento inoltre grande flessibilità, potenza e capacità di orchestrazione.

[4] https://azure.microsoft.com/it-it/

[5] https://aws.amazon.com/it/?nc2=h_lg

[6] https://cloud.google.com/why-google-cloud?hl=it

A completare il design architetturale del modulo Fornitore di servizi, analizziamo la vNSF insieme agli scenari di configurazione che quest'ultima dovrà supportare.
La vNSF, in questo caso il WAF che dovrà essere istanziato, deve riuscire a proteggere un generico servizio web reaggendo a diverse tipologie di attacco, nonchè riuscire a fornire un adeguato supporto alla gestione di questo servizio; in particolare, si introducono alcuni dei principali casi d'utilizzo della soluzione, selezionati in fase di progetto:

1. l'istanza del WAF deve garantire la protezione dalle principali tipologie di attacchi descritti all'interno dell'*OWASP Top Ten*;
2. l'istanza del WAF deve garantire la protezione da programmi di scripting, di bot e di crawler;
3. l'istanza del WAF deve poter garantire restrizioni sugli accessi verso un determinato Fully Qualified Domain Name (FQDN);
4. l'istanza del WAF deve poter restringere il traffico consentito a quello proveniente da un determinato paese;
5. l'istanza del WAF deve poter garantire la creazione di file di Log, facilmente consultabili ed analizzabili, anche in relazione al livello di sicurezza che si sta applicando.

Per le ragioni sopra esposte, si è deciso di utilizzare una versione di ModSecurity WAF, installato come plugin del web server Nginx all'interno di un container Docker. Nonostante kubernetes implementi nativamente una versione di ModSecurity-Nginx, si è optato per una configurazione ed implementazione exnovo, considerando la vNSF come struttura assestante ed indipendente dalla piattaforma su cui viene eseguita. Di conseguenza, il WAF viene istanziato in modalità enbedded seguendo l'archhitettura di tipo inline introdotta nel Capitolo 5.
Naturalmente ModSecurity, come Kubernetes, è stato scelto e preferito agli altri competitor open source per via della sua grande diffusione, del grande supporto offerto dalla community e dall'alto rendimento prestazionale.

6.4 Flusso di lavoro

Nelle precedenti Sezioni, di questo Capitolo, abbiamo analizzato le funzionalità dei moduli logici facenti parte dell'architettura, in questa Sezione si analizzerà il flusso completo di lavoro operato dalla soluzione, per fornire un'immagine più chiara dell'interazione dei moduli.

In Figura 6.5 viene rappresentato il diagramma di sequenza delle interazioni tra le varie componenti dei moduli logici, in particolare si approfondisce l'interazione tra il framework NEMO e l'Orchestratore.
Il processo di istanziazione delle vNSF viene scatenato dall'interazione tra utente ed orchestratore, rappresentati in figura come un'unica entità. L'utente descrive

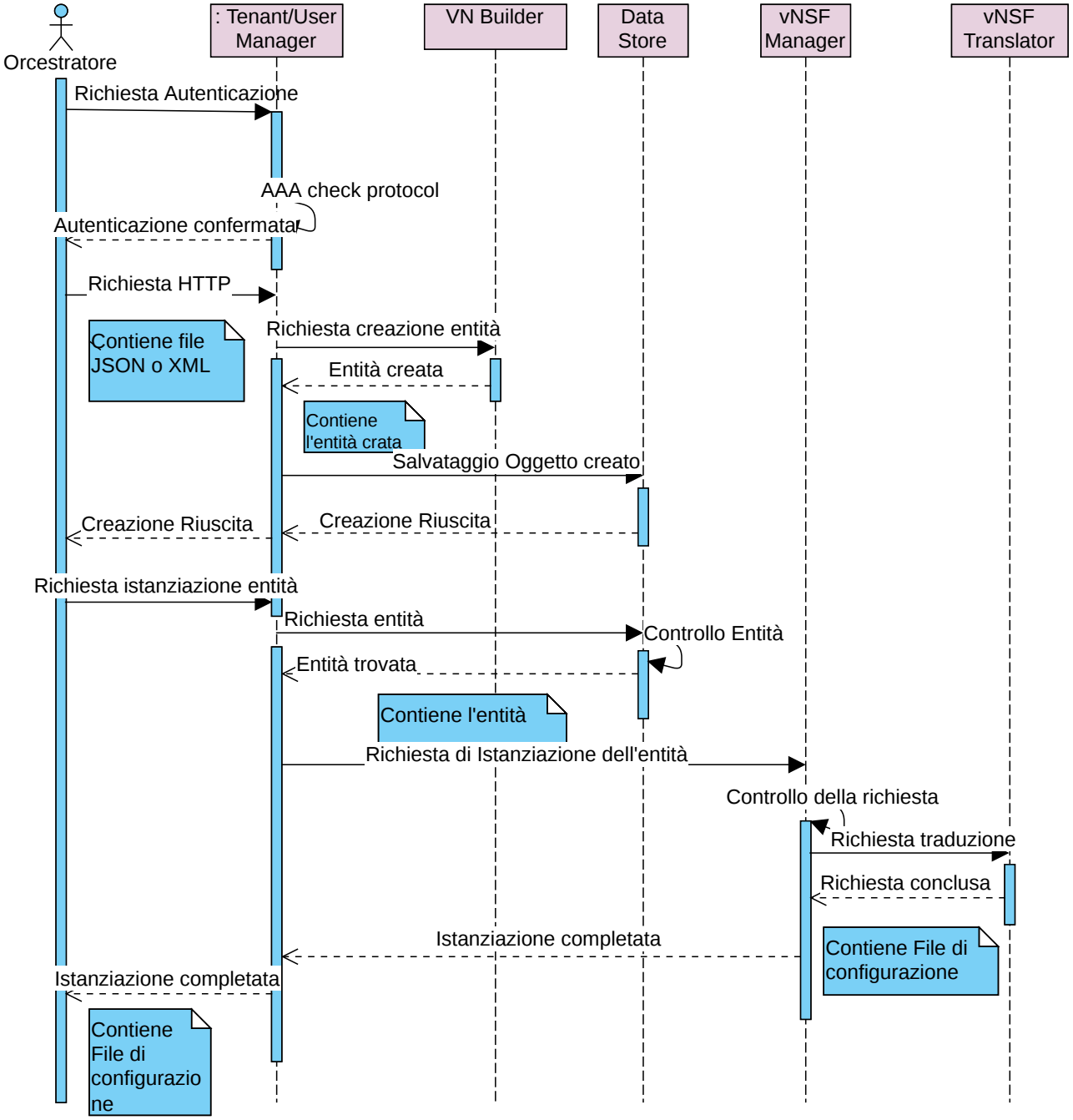


Figura 6.5: Diagramma di Sequenza per la creazione del file di configurazione.

all'orchestratore l'intento riferito alla vNSF che vuole istanziare e, tramite una richiesta HTTP verso l'interfaccia REST del framwork NEMO, quest'ultimo inizia il processo di salvataggio.
All'interno di NEMO lo User Manager intercetta il messaggio ricevuto dall'orchestratore, la prima operazione che compiono riguarda l'autenticazione dell'utente, in quanto, come descritto precedentemente, coincide con il fulcro di tutta la topologia di rete che dovrà essere salvata. Dopo che l'autenticazione è andata a buon fine l'orchestratore invia a NEMO, sottoforma di file JSON o XML, l'intento che dovrà essere scomposto in entità, proprietà e connessioni per poi essere salvato all'interno

del Data Store. La prima interazione si conclude con un messaggio di "salvataggio effettuato", inviato come feedback all'utente finale. A questo punto, l'utente può decidere di chiedere l'implementazione effettiva della vNSF definita poc'anzi. Per effettuare l'implementazione, l'utente invoca l'orchestratore che effettua due operazioni: in primo luogo contatta nuovamente NEMO per chiedere la generazione del file di configurazione, dopo di che effettua la traduzione del file ricevuto ed apre una connessione SSH, per configurare la funzione sulla piattaforma che gli è stata indicata.
La nuova interazione tra orchestratore e NEMO viene sempre intercettata dallo User Manager, il quale controlla nel data store l'entità che si vuole istanziare ed affida il compito di generare il file di configurazione al vNSF Manager. Il vNSF Manager controlla la validità della richiesta e la tipologia di entità ricevuta per invocare, a sua volta, la componente vNSF Translator.
NEMO consegna il file di configurazione creato all'orchestratore, il quale traduce il file di configurazione generico in delle configurazioni di basso livello per poi interaggire con Kubernetes, completando lo scenario implementativo.
L'interazione tra kubernetes e l'orchestratore avviene tramite una singola sessione di comunicazione, infatti, dopo che le configurazioni a basso livello sono state create si apre una sessione SSH per inviarle ed istanziare il container docker di ModSecurity-Nginx opportunamente configurato.

Si ritiene opportuno non discutere di flussi alternativi come la modifica di un'entità già presente all'interno del framework, dal momento che l'interazione tra i moduli non subirebbe variazioni.

Capitolo 7

Implementazione della soluzione

In questo Capitolo si analizzerà l'implementazione dell'architettura dello scenario proposto da questa tesi, introdotto nel Capitolo 6.
Per comprendere come sia organizzato lo scenario implementativo si può analizzare la Figura 7.1, che riporta l'architettura ad alto livello della Sezione 6.1, dove vengono evidenziati i moduli che sono stati revisionati o realizzati exnovo. Dalla Figura,

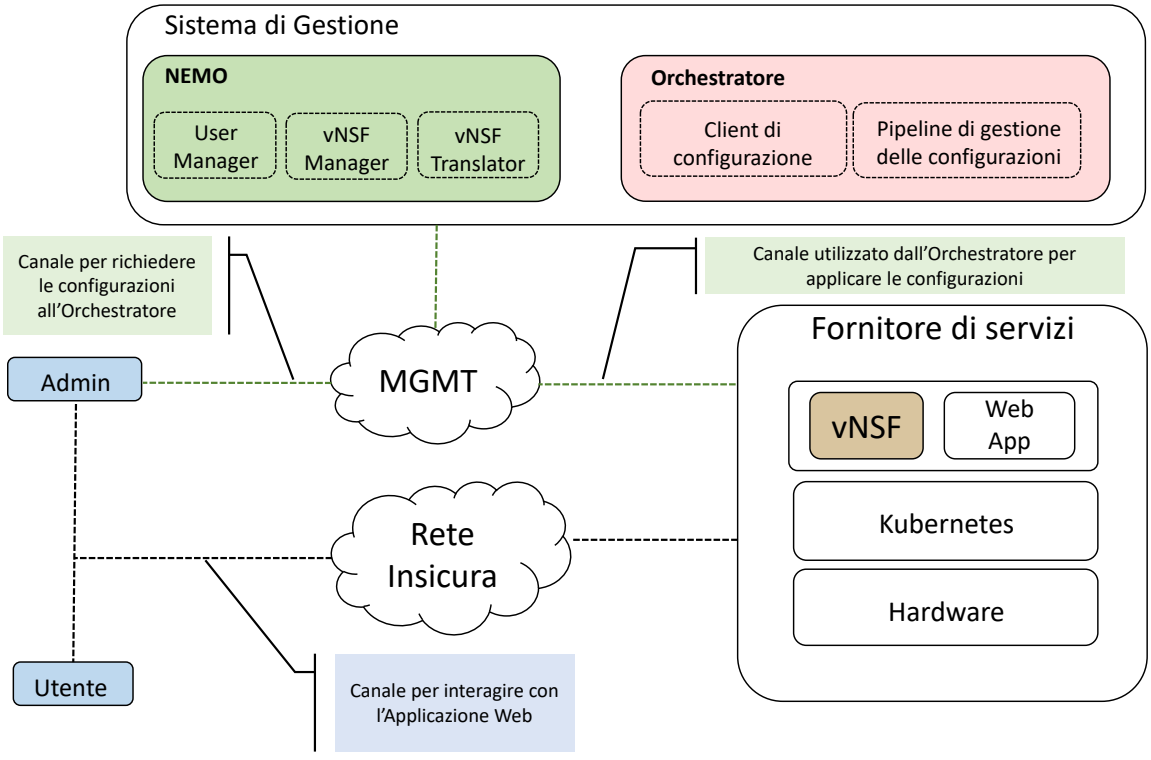


Figura 7.1: Architettura implementativa ad alto livello.

si evince che il modulo del Sistema di Gestione rappresenta il cuore del processo implementativo della soluzione, in quanto nel modulo del Fornitore di servizi le uniche modifiche che vengono apportate riguardano le configurazioni del WAF.
I moduli della piattaforma NEMO e dell'Orchestratore, appartenenti al Sistema di Gestione, sono stati implementati all'interno del medesimo sistema virtualizzato, a

differenza di Kubernetes, rappresentante del Fornitore di servizi, che viene installato in un differente spazio virtualizzato; questa separazione vuole evidenziare la suddivisione in ruoli creata in fase di progettazione. Idealmente tutti i moduli funzionali possono essere implementati su ambienti virtualizzati differenti, su macchine fisiche distinte o all'interno dello stesso sistema, tuttavia lo scopo della soluzione è creare dei compartimenti, ben separati, assegnabili a diversi attori del mondo reale. Un esmpio potrebbe essere quello di assegnare la piattaforma NFV, come Kubernetes, ad una delle più note aziende di IaaS e di assegnare il Sistema di Gestione ad un'azienda capace di gestire, tramite intenti, le suddette piattaforme.

Nel prosieguo di questo capitolo si analizzeranno nel dettaglio le modifiche effettuate su NEMO, la realizzazione dell'Orchestratore e, per finire, ci concentreremo sia su come è stato istanziato Kubernetes, sia su come è stata creata la vNSF al suo interno.

7.1 Modifica nel Framework NEMO

Di NEMO, fin'ora, sono state definite le caratteristiche generali e le funzionalità dei principali moduli, analizzando, in fase di progettazione, le possibili modifiche per adattarlo alla soluzione proposta da questo lavoro.
Prima di approfondire quali cambiamenti sono stati applicati alla piattaforma NEMO, è doveroso esplicitare l'idea su cui si basa l'intero pregetto, approfondendo, in particolar modo, la chiave di lettura delle modifiche apportate. L'idea di base è quella di valutare il riutilizzo completo della piattaforma OpenDelight NEMO, cercando di adattare l'intero set di regole, già presente al suo interno, alla soluzione proposta, valutando piccole modifiche per migliorarne utilizzabilità e prestazioni. Si adotta quindi un'approccio molto conservativo per valutare la flessibilità, l'adattabilità e le prestazioni di piattaforma e linguaggio. Per queste ragioni, al fine di comprendere al meglio gli interventi da effettuare, si rende necessario ridefinire le richieste implementative espresse durante la progettazione: in primo luogo troviamo l'esigenza di offrire all'utente, che si relaziona con la piattaforma, un'interfaccia capace di accettare intenti contenenti le proprietà relative ad una vNSF; in secondo luogo dobbiamo modificare l'output della piattaforma per farlo aderire allo scenario richiesto, definito nella Sezione 6.1. Gli interventi che ci hanno consentito di ottenere le caratteristiche sopra esposte, riguardano l'ampliamento della sintassi di NEMO e la realizzazione del modulo di traduzione.

7.1.1 Estensione della sintassi

L'ampliamento della sintassi di NEMO è finalizzata alla definizione di intenti completi ed interpretabili correttamente. Le modifiche effettuate nascono dai concetti e dalle metodologie esaminate nella Sezione 4.2.2, a cui faremo riferimento più volte all'interno di questa sottosezione.

La prima modifica che si vuole effettuare riguarda l'aggiunta dell'entità WAF all'interno della piattaforma NEMO; l'obiettivo è quello di rendere noto alla piattaforma l'esistenza del WAF come funzione di sicurezza, specificando le proprietà

caratterizzanti della vNSF e le tipologie di interfacce che può supportare. Per effettuare tale implementazione, si effettua una modifica sul file di configurazione delle entità che NEMO accetta in ingresso. Come mostra la Figura 4.8, questo file verrà utilizzato da modello per la struttura dati in cui conservare le entità, con le relative proprietà, specificate dall'utente tramite intenti.
Nel Listato 7.1, partendo dalla struttura *Nodo*, appartenente all'entità *Object* definita nel linguaggio IB-NEMO (Figura 4.5), si caratterizza la vNSF specificando la nuova tipologia del nodo e definendo le sue nuove proprietà:

```
{
    "node-type": "waf",
    "property-definition": [
        {
            "property-name": "application_to_protect",
            "property-value-type": "string",
            "is-required": "required"
        },
        {
            "property-name": "active_protection_attack",
            "property-value-type": "string"
        },
        {
            "property-name": "block_domain_name",
            "property-value-type": "string"
        },
        {
            "property-name": "block_bad_user_agent",
            "property-value-type": "string"
        },
        {
            "property-name": "block_ip_class",
            "property-value-type": "string"
        },
        .
        .
        .
        {
            "property-name": "service_port",
            "property-value-type": "int",
            "is-required": "required"
        },
        {
            "property-name": "ip-address",
            "property-value-type": "string"
        }
    ]
}
```

Listing 7.1: Modifica del File di configurazione NEMO.

Le proprietà del nuovo nodo, definito come tipologia "*waf*", possono essere sia obbligatorie che opzionali, consentendo all'utente l'utilizzo degli intenti in una duplice modalità. Le tre proprietà considerate obbligatorie, marcate con il valore *required*, ci permettono l'istanziazione del WAF senza che l'utente sia consapevole di come esso operi e quali tipologie di protezione offra; infatti l'utente potrà specificare l'obiettivo di instaziare, a protezione di un dato servizio Web, un WAF senza essere a conoscenza di come queste operazioni vengano effettuate, rispettando, così, i principi dell'interazione basata sugli intenti specificati nel Capitolo 3. La seconda modalità di utilizzo permette, ad un utente più consapevole, non solo di definire il livello di protezione da dare al servizio Web, ma anche di specificare da che tipologie di attacchi il WAF debba proteggerci, dichiarandoli esplicitamente.

Per comprendere al meglio come un utente possa istanziare il WAF, in riferimento alla sintassi della Sezione 4.2.1 e alle modifiche appena definite, si mostrano dei possibili intenti accettabili dalla piattaforma NEMO:

```
1 #Prima modalita' di utilizzo degli intenti
2 CREATE Node nextCloud Type waf Property application_to_protect:
      nextcloud, alert_level: medium, service_port: 8080;
3
4 #Seconda modalita' di utilizzo degli intenti
5 CREATE Node nextCloud Type waf Property application_to_protect:
      nextcloud, alert_level: higth, service_port: 80,
      active_protection_attack: xss-sql-dos-php-java-rce-ip-shell,
      block_bad_user_agent: bot-scanner, block_country_traffic: it;
```

Il primo intento, riportato nell'esempio, richiede la creazione di un WAF con un livello di protezione medio posto a protezione del servizio web "nextcloud", esposto sulla porta 8080. Come anticipato, l'utente che utilizza la prima tipologia di intento non ha bisogno di conoscere i pericoli in cui un servizio Web possa imbattersi.
Il secondo intento, invece, mostra delle caratteristiche molto differenti rispetto al primo, invero, qui, l'utente deve essere consapevole dei pericoli che un servizio Web possa incontrare, dal momento che deve dichiarare esplicitamente le tipologie di protezione che si vogliono adottare. Le tipologie di protezione previste in questo particolare lavoro fanno riferimento a quelle specificate in fase di progettazione (Sezione 6.3.1).

Come anticipato nella prefazione di questa Sezione, le entità definite nella piattaforma sono completamente riutilizzate, infatti si vuole porre particolare attenzione sulla proprietà "active_protection_attack" di tipo "string"; questa proprietà, specificata all'interno dell'esempio, riporta una combinazione di parole separate dal carattere "-", configurando, di fatto, un'unica stringa come un insieme di termini al fine di creare una lista. Tale soluzione consente il riutilizzo, seppur in modo poco elegante, delle strutture già presenti all'interno della piattafarma.
L'implementazione exnovo di una struttura lista all'interno del framework è stata scartata fondamentalmente per due ragioni, la prima riguarda l'assenza di guadagno in termini prestazionali dovuti alla modifica implementativa, la seconda, invece, si concentra sulla tipologia di modifica che sarebbe dovuta essere applicata alla piattaforma. Di fatto una modifica di questo genere avrebbe portato alla rettifica totale

di tutte le entità presenti nel modulo MD-SAL, del parser JavaCC e di tutti i moduli strutturali che si occupano dell'immagazzinamento delle informazioni all'interno del Data Store, senza apportare alcuna miglioria.
Malgrado l'assenza di modifiche strutturali, NEMO ha consentito ugualmente di implementare questo tipo di entità senza subire cali di prestazioni o problemi di funzionamento, evidenziando la sua grande flessibilità ed adattamento a nuove strutture e proprietà. (Maggiori dettagli nell'Appendice B).

7.1.2 Implementazione del modulo di Traduzione

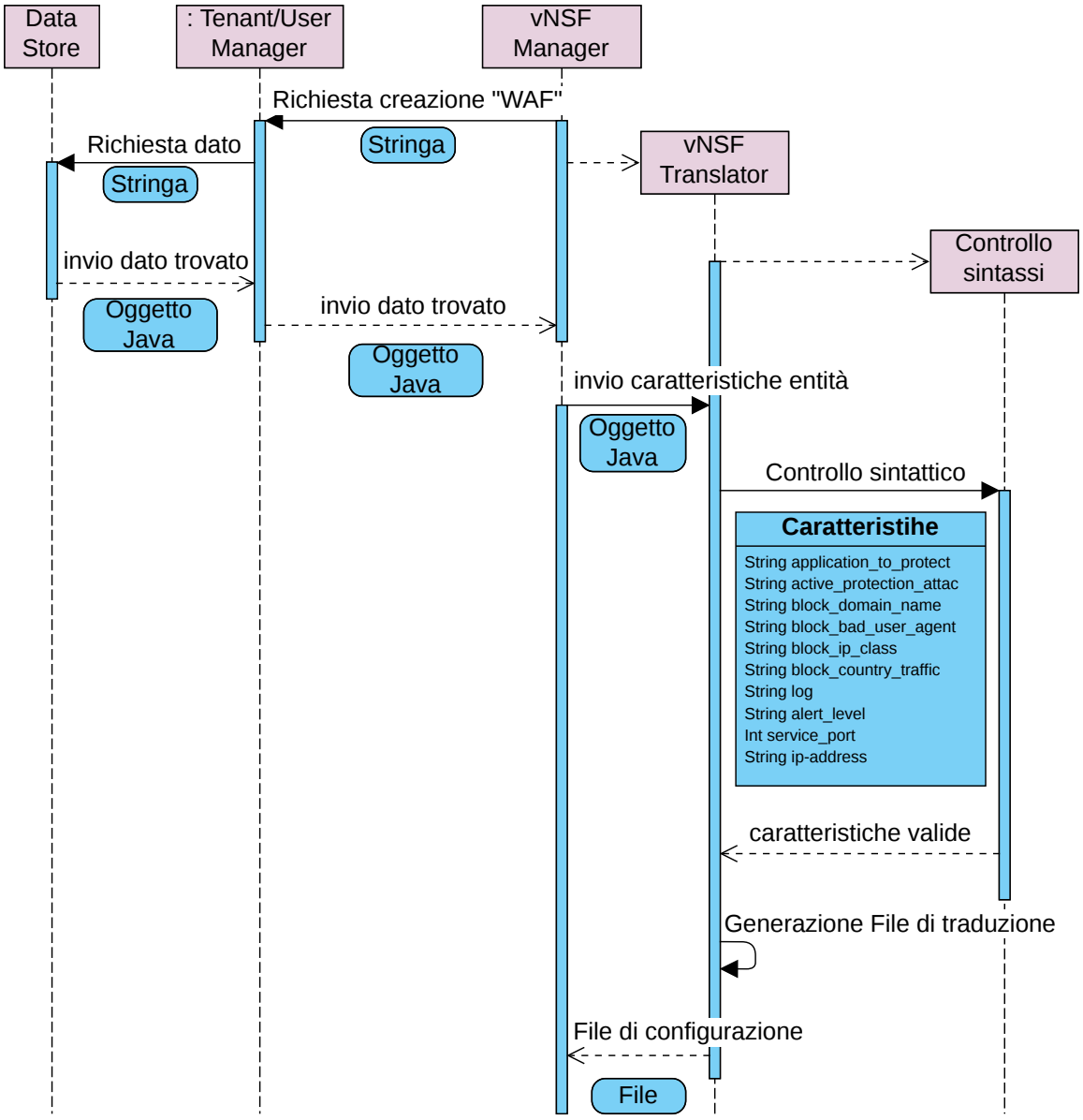


Figura 7.2: Flusso operativo dei moduli di traduzione.

La modifica dell'output della piattaforma, definita nella Sezione 6.2.1, prevede l'accantonamento del modulo *Physical Network Manager* in favore dei moduli *vNSF*

Manager e *vNSF Translator*.

Al fine di comprendere al meglio dove e come sono implementate le modifiche dell'output della piattaforma, si puo analizzare la Figura 7.2, specializzazione della Figura 6.5, che riporta il flusso operattivo dei nuovi moduli in relazione alle tipologie di informazioni che vengono scambiate. Dalla figura si evince che il vNSF Manager si comporta da intermediario tra il Tenant Manager e il vNSF Translator, gestendo, di fatto, l'istanziazione del modulo di traduzione e la realizzazione del file di configurazione.
Il vNSF Translator riceve in ingresso l'oggetto rappresentante l'entità salvata nel Data Store, ne estrae le proprietà e, dopo aver verificato la correttezza sintattica tramite il modulo di controllo della sintassi, genera un file di configurazione che sarà consegnato all'utente che ne ha effettuato richiesta.
Il file di configurazione, generato dai moduli sopra indicati, risulta essere una rappresentazione ad alto livello delle configurazioni che dovranno essere implementate su una piattaforma NFV. La sintassi del file è fortemente legata alle caratteristiche che la funzione di sicurezza possiede, creando una sintassi semplice, intuitiva e flessibile, capace di discriminare quali configurazioni vadano attivate e quali no; sarà compito dell'orchestratore tradurre queste configurazioni di alto livello in quelle di basso livello.

L'implementazione effettiva del modulo di traduzione avviene estendendo le classi Java già presenti nel NEMO engine, introducendo il vNSF Manager e vNSF Translator come due classi seprate. L'oganizzazione della progettazione e dell'implementazione sono state definite in modo da garantire una grande flessibilità, qualora si decidesse di estendere il supporto ad un insieme di fuznioni di rete più ampio.

7.2 Orchestratore

In questa sezione definiamo l'implementazione del modulo *Orchestratore*, introdotto in fase di progettazione nella Sezione 6.2.2, che si occupa di gestire la comunicazione tra Utente, piattaforma di gestione degli intenti e piattaforma NFV.

Come riportato in Figura 6.3 questo modulo è suddiviso in due moduli Logici, rispettivamente *Client di configurazione* e *Pipeline di gestione delle configurazioni*. Il primo modulo logico a sua volta si divide in tre componenti funzionali, utilizzate in momenti diversi durante l'instanziazione delle configurazioni. Queste componenti sono utilizzate dalla Pipeline di gestione delle configurazioni, che li richiama insieme a specifiche librerie utili per il completamento del lavoro.
Per comprendere in toto il funzionamento dell'orchestratore si analizzerano singolarmente tutti i moduli da cui e costituito:

- *Modulo di configurazione di NEMO*: in questo modulo si implementano tutti i metodi per effettuare la prima configuraizone di NEMO, infatti, grazie all'interfaccia REST esposta sulla porta 8181, è possibile modellare un file JSON contenente tutte le entità che la piattaforma utilizza per costruire la topologia virtuale. Come riportato nella Sezione 7.1.1, questo file contiene una descrizione dettagliata di tutte le entità, con le relative caratteristiche, che NEMO

deve essere in grado di comprendere e registrare. Il Listato 7.1, che si riferisce alla definizione del nodo WAF, è implementato in questo modulo. Come già detto, questo file ci consente un'estrema flessibilità nella modellazione della sintassi, in quanto una modifica alle entità JSON, ivi riportate, si tramuta in una "nuova" sintassi per la piattaforma.

- *Modulo di generazione intenti NEMO*: in questo secondo modulo, similarmente al primo, si interaggisce ancora con l'interfaccia RESTful della piattaforma, ma utilizzando i metodi "*operations*" che permettono all'utente una diversa interazione. Di fatto, questo modulo, simula l'interazione tra Utente e NEMO permettendo la definizione di intenti, la registrazione di nuovi utenti e l'implementazione delle configurazioni riferite alla topologia di rete salvata all'interno della piattaforma stessa. In aggiunta, questa interfaccia rispetta le peculiarità del modulo MD-SAL aggiungendo un metodo per l'inizio e uno per la chiusura di una transazione.
  Tutti i metodi offerti dall'interfaccia REST devono essere eseguiti dichiarando una transazione su uno specifico utente, contattando, tramite richiesta *POST*, l'opportuno indirizzo.

- *Modulo di taduzione e configurazione NFV*: il modulo di traduzione risulta essere una componente fondamentale per l'intero scenario implementativo, infatti prendendo in ingresso un file generico di configurazione, elaborato dal framework NEMO, riesce a tradurlo in configurazioni di basso livello instanziabili all'interno di Kubernets.
  Questo modulo viene implementatato in Python e, per effettuare la traduzione, si avvale fondamentalmente di due elementi chiave: il primo si identifica in dei template di configurazione precompilati, selezionabili in base allo scenario delineato dal file di configurazione generico preso in ingresso, il secondo elemento consiste nella possibilità di disaccoppiamento tra vNSF selezionata e piattaforma NFV. In sostanza, tramite l'utilizzo di jinja[1], come configuratore di tamplate, possiamo effettuare rapidi cambi di configurazione, modificano i template che vengono forniti al modulo, per instanziare WAF differenti sulla stessa piattaforma NFV, senza toccare nemmeno una riga di codice.
  A questo punto, si può effettuare un paralellismo tra questo modulo e il modulo di Renderer presente all'interno di NEMO, in quanto entrambi svolgono la medesima funzione, ovvero si occupano di implementare le funzioni di rete, conservate all'interno del framework, mappando le entità ricevute dal modulo di gestione in configurazioni di basso livello.

- *Pipeline di gestione delle configurazioni*: quest'ultimo modulo si occupa di gestire le comunicazioni che avvengono tra le componenti funzionali e i principali attori dello scenario implementativo. In particolare, oltre a gestire gli altri moduli, si occupa della comunicazione tra Modulo di taduzione e piattaforma NFV aprendo un canale SSH e comunicando le configurazioni in maniera

[1] https://jinja.palletsprojects.com/en/3.1.x/

opportuna. Per effettuare tali comunicazioni si avvale della libreria *paramiko*[2], che permette l'apertura e la gestione di comunicazioni SSH, delle librerie *HTTPBasicAuth*[3] e *requests*[4], per gestire sessioni e comunicazioni HTTP con l'interfaccia REST, e della libreria JSON per modellare gli omonimi file in fase di configurazione.

Come evidenziato nella descrizione appena effettuata delle componenti, l'Orcherstratore ricopre un ruolo non convenzionale all'interno del sistema, permettendo un'automazione nell'istanziazione delle configurazioni riferite alle vNSF, all'interno dello scenario implementativo. L'automazione dello scenario consente di valutare sia le prestazioni, che l'usabilità della soluzione proposta, che saranno valutate nei successivi capitoli.

7.3 Fornitore di servizi

Il Fornitore di servizi, introdotto nella Sezione 6.3, è implementato mediante software ampiamente utilizzato nel mercato del Cloud computing, selezionando Kubernetes per la piattaforma NFV e ModSecurity per l'implementazione del WAF. In questa sezione andremo ad analizzare dettagliatamente come sono stati instanziate e configurate le componenti del Fornitore di servizi.

7.3.1 Piattaforma NFV

Per configurare l'ambiente operativo di Kubernetes, all'interno della soluzione, si sono dovute affrontare diverse problematiche, in quanto la piattaforma di orchestrazione richiede dell'hardware dedicato per permettere l'istanziazione di un *cluster* di lavoro, composto da almeno un nodo *master* e due *worker*, come spiegato nella Sezione 2.5.1. Oltre alla problematica dell'istallazione e della configurazione iniziale su hardware dedicato, insorge l'incognita del tipo di CNI da utilizzare per garantire la comunicazione tra tutte le componenti base che compongono kubernetes stesso. Per queste ragioni si è scelta la piattaforma *Minikube* [58], creata dalla stessa Google, capace di fornire un'esecuzione locale dell'orchestratore Kubernetes, sfruttando risorse virtualizzate limitate. Nello specifico Minikube, per il suo funzionamento, richiede almeno due CPU virtuali, 2GB di RAM, 20GB di spazio libero sul disco e un gestore di container o di macchine virtuali compatibile.
Grazie alla piattaforma Minikube non si è reso necessario nè l'utilizzo di server dedicati o di servizi di cloud particolari, nè di astruse configurazioni iniziali o plugin di terze parti per effettuare l'implementazione della soluzione proposta da questo lavoro.

[2] https://www.paramiko.org/

[3] https://requests.readthedocs.io/en/latest/user/authentication/

[4] https://requests.readthedocs.io/en/latest/

Minikube e Kubernetes condividono la medisima architettura, le uniche differenze consistono nella configurazione iniziale e nel fatto che Minikube venga inizializzato all'interno di una macchina virtuale. La versione vanilla di Minikube prevede l'istanziazione di un solo nodo master ed un sono nodo worker, monta una CNI basica, Kindnet, che non consente complesse configurazioni di rete, ma permette un utilizzo completo dell'orchestratore.
Minikube, per lo scenario della soluzione proposta, è configurato come macchina virtuale all'interno del sistema operativo Windows 10, sfrutta Kindnet come CNI e utilizza Docker come gestore di container. Le impostazioni relative alla macchina virtuale e l'hardware utilizzato saranno approfondite nel Capitolo 8, quando si andranno a valutare le performance dell'intero sistema.

7.3.2 ModSecurity vNSF

Il software utilizzato per implementare il WAF è ModSecurity, la cui istanziazione è prevista all'interno di un container Docker, come modulo del Web server Nginx. Per comprendere appieno le funzionalità e l'architettura di ModSecurity, in riferimento ai casi d'utilizzo introdotti nella Sezione 6.3.1, si analizzeranno, nel dettaglio, l'implementazione del modulo all'interno di Nginx, valutando tutte le configurazioni necesarie al raggiungimento degli obiettivi proposti.

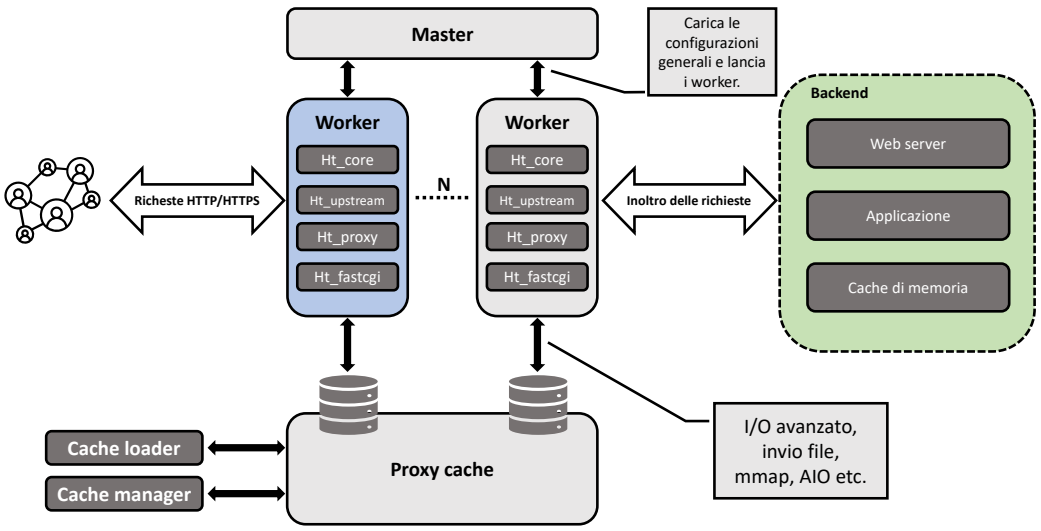


Figura 7.3: Architettura di Nginx ad alto livello[5]

Nginx [59] è un Web server che nasce in risposta alle esigenze emerse con l'avvento di internet su larga scala, infatti, insorgeva l'esigenza, per molte applicazioni web, di un sistema che permettesse di istanziare applicazioni scalabili, con la possibilità di gestire una moltitudine di connessioni e funzionalità senza rinunciare alle

[5]La reference a questa immagine si può trovare al seguente url https://www.aosabook.org/en/nginx.html#sec.nginx.internals

performance. Le funzionalità offerte da questa piattaforma sono molteplici, tra cui sistemi per il bilanciamento del carico o di reverse proxy, quest'ultimo è particolarmente interessante, al completamento della soluzione, in quanto permette l'utilizzo del software ModSecurity.
Nginx, quindi, prevede un'architettura modulare che permette l'istanziazione di componenti chiamate *Worker*, capaci di intercettare una moltitudine di connessioni, gestendo al meglio le risorse hardware a disposizione.

In Figura 7.3 viene rappresentata, ad alto livello, l'architettura modulare di Nginx, qui si può osservare come il nucleo operativo sia concentrato tutto sulle componenti definite worker; queste componenti si occupano della gestione delle funzioni di *core* e dell'invocazione dei *moduli funzionali*. Le funzioni di core si occupano di creare un Loop di gestione delle richieste altamente efficiente, in grado di chiamare le opportune funzioni quando risulta necessario, il tutto senza sprecare cicli di CPU. Per quanto riguarda i moduli funzionali, questi vengono invocati dal core per elaborare le richieste ricevute ed inoltrarle nel backend a livello applicativo, leggendo e scrivendo informazioni sulla rete e sullo storage, trasformandone il contenuto, e applicando azioni di filtraggio dei messaggi sia in entrata che in uscita del canale di comunicazione.
Questa architettura modulare consente agli sviluppatori di estendere l'insieme delle funzionalità del Web server, senza modificare il nucleo operativo di nginx. I moduli di nginx possono essere instanziati in punti del sistema molto differenti, generando, di fatto, un insieme di moduli eterogeneo, ovvero moduli del core, moduli basati su eventi, gestori di fase, protocolli, gestori di variabili, filtri e bilanciatori di carico.
Nginx pur avendo una struttura modulare necessita di essere compilato exnovo ogni volta che uno spivuppatore vuole aggiungere una nova funzionalità, caricando nel sistema il modulo funzionale oppurtono, per poter poi essere usato dal worker in fase di runtime.
Per le ragioni sopra esposte, ModSecurity è implementato come modulo funzionale di nginx, di conseguenza compilato con il web server, sotto forma di funzioni realizzate in linguaggio C che vengono invocate da un worker ogni volta che si riceve una richiesta HTTP.

Gli obiettivi implementativi, definiti nella Sezione 6.3.1, sono stati raggiunti configurando il modulo ModSecurity con una versione modificata delle OWASP Core Rule Set, aggiungendo nuove regole tramite il linguaggio SecRule ed aggiungendo un ulteriore modulo in Nginx, per permettere di identificare l'origine geografica delle richieste elaborate.
Il modulo utilizzato in concomitanza con ModSecurity è GeoIP[6], il quale consente, tramite l'analisi di database preconfigurati, l'individuazione geografica dell'indirizzo IP che effettua una richiesta verso il Web server. La principale problematica riscontrata nell'utilizzo del modulo GeoIP riguarda lo scarso supporto ai database preconfigurati da dare in pasto al modulo, le uniche opzioni valide, aggiornate ed affidabili risultano essere a pagamento, nonostante questo all'interno dello scenario proposto si utilizza una versione lite gratuita di questi database.

[6] https://docs.nginx.com/nginx/admin-guide/dynamic-modules/geoip/

Capitolo 8

Risultati

Si premette che la soluzione proposta da questo lavoro, ampiamente discussa nei Capitoli precedenti, mira ad ottenere una semplificazione della gestione dei servizi di sicurezza, applicati alla rete da parte dell'utente. Per questa ragione, risulta complicato quantificare i risultati che si possono ottenere da questo scenario prototipale, tuttavia è fondamentale fornire un Proof-of-Concept (PoC[1]) del sistema analizzandone le prestazioni e la fattibilità.
In questo Capitolo, quindi, si andranno ad analizzare i risultati operativi, in termini di performance, prodotti dall'intero scenario implementativo, prendendo come dati di riferimento le tempistiche necessarie al completamento di un intero ciclo di configurazioni da parte del sistema, eseguendo le opportune comparazioni e valutazioni.

8.1 Ambiente operativo

Per valutare, in prospettiva di un'applicazione reale del prototipo, la bontà dei test sperimentali effettuati, risulta di fondamentale importanza descrivere brevemente l'ambiente operativo su cui questi vengono eseguiti.

In Figura 8.1 viene rappresentata l'architettura dello scenario operativo introdotto nella Prefazione del Capitolo 7, dove si evidenzia l'utilizzo di una sola macchia fisica definita di Managment (MGMT) in cui vengono eseguiti Orchestratore, piattaforma NEMO e Kubernetes. Come evidenziato nella Prefazione, tutti gli strumenti utilizzati nello scenario sono eseguiti in macchine virtuali distinte, ciascuna con caratteristiche differenti.
Naturalmente, l'implementazione proposta non rappresenta l'ambiente operativo ideale, in quanto si è costretti a sfruttare pesantemente la virtualizzazione per implementare le singole componenti, inficiando molto sulle prestazioni globali del sistema.
Un possibile ambiente operativo ideale potrebbe includere due sistemi distribuiti,

[1]Il termine è ampiamente usato in ambito informatico, si riferisce alla dimostrazione pratica dei funzionamenti di base di un applicativo software o di un intero sistema, integrandolo all'interno di un ambiente già esistente.

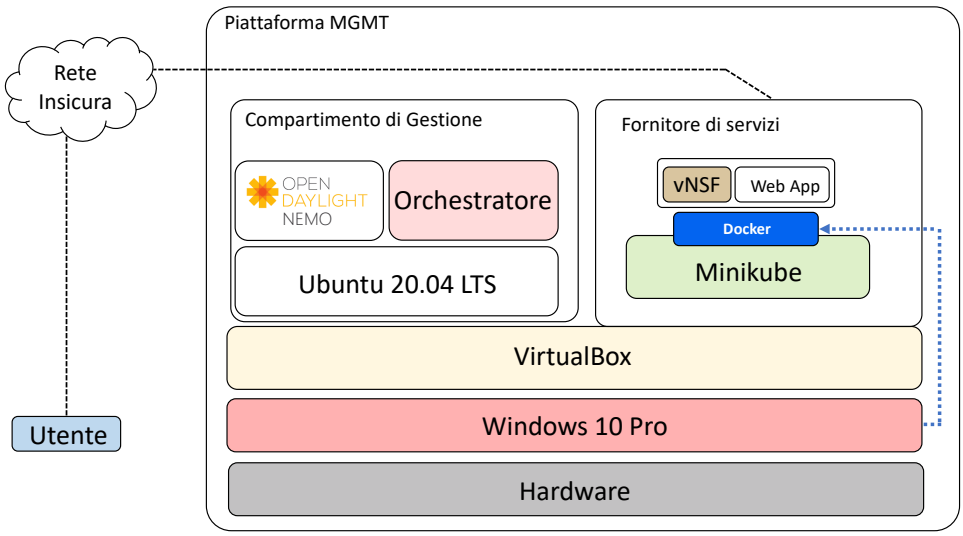


Figura 8.1: Architettura dello scenario operativo.

uno per la piattaforma NFV ed uno relativo all'orchestratore, il quale si pone come Layer adattivo di comunicazione verso la piattaforma NEMO; così facendo, il carico di lavoro risulterebbe decentralizzato rispetto l'unità computazionale dell'utente, favorendo del boost di performance fornito dagli operatori di cloud computing, o dalle unità computazionali dedicate. Tale possibilità può essere intesa come un possibile ampliamento futuro del prototipo, che non inficia sullo studio di fattibilità e performance dello scenario attuale.

Le caratteristiche di ogni singola componente presente nell'architettura sono:

- *Unità di MGMT*:
  - Scheda Madre: Huananzhi x79, chipset c600, supporta standard socket LGA2011, PCIe 3.0;
  - CPU: Intel(R) Xeon(R) CPU E5-2640 v2[2] @ 2.00GHz, 8 cores, 16 processori logici;
  - RAM: 16GB DDR3 ECC @ 1866MHz;
  - Storage: HDD meccanico sata 3, 5000rpm, da 465GB;
  - Sistema Operativo: Windows 10 pro v21H2;
- *Piattaforma del Sistema di Gestione*:
  - CPU: CPU Virtuale @ 2.00GHz, 4 cores;
  - RAM: 6GB DDR3 ECC @ 1866MHz;
  - Storage: HDD VBox (unità virtuale) da 25GB;

[2] https://ark.intel.com/content/www/it/it/ark/products/75267/intel-xeon-processor-e52640-v2-20m-cache-2-00-ghz.html

  - Sistema Operativo: Ubuntu 20.04 LTS;

- *Sistema del fornitore di servizi*:
  - CPU: CPU Virtuale @ 2.00GHz, 2 cores;
  - RAM: 2GB DDR3 ECC @ 1866MHz;
  - Storage: HDD VBox (unità virtuale) da 20GB;
  - Sistema Operativo: Minikube;

8.2 Analisi e test

In questa sezione si andranno a descrivere le tipologie di test che sono stati effettuati, in relazione al parametro tempo utilizzato per per valutare le performance.

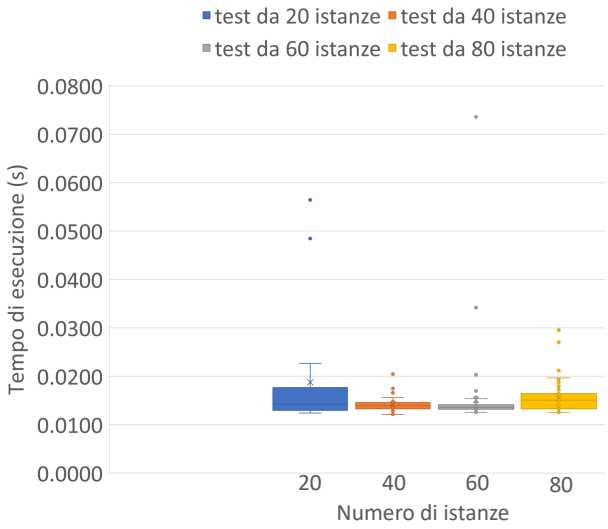
8.2.1 Moduli NEMO

Le modifiche effettuate alla piattaforma NEMO rappresentano il target dei test proposti in questa Sezione, i quali hanno il compito di stressare la piattaforma per comprendere quanto questa sia affidabile e performante. Lo scenario consiste nell'istanziazione virtuale, da parte di un utente, di un numero crescente di servizi di sicurezza, con caratteristiche differenti, e nella richiesta di generazione del relativo file di configurazione. In particolare si effettuano quattro test differenti, richiedendo, rispettivamente, la generazione di 20, 40, 60 e 80 WAF virtuali con relativo file di configurazione.
Per automatizzare la simulazione si riutilizza, modificandolo leggermente, il modulo di generazione degli intenti presente nell'Orchestratore, il quale rientrerà nell'analisi delle performance, in quanto parte attiva della simulazione, capace di fornire un feedback delle operazioni all'utente. La modifica del modulo, di generazione degli intenti, consiste nell'eliminazione della fase di registrazione di un utente e nella possibilità di comunicare una moltitudine di intenti e richieste di istanziazione, anzichè limitarsi ad una sola interazione.
Gli intenti utilizzati, nella creazione di un singolo WAF, possiedono caratteristiche differenti in modo da valutare quanto vari il carico di lavoro in base a piccole e grandi variazioni nella configurazione. Si specifica che gli intenti sono basati sulle caratteristiche del WAF introdotte nella Sezione 7.1.1.

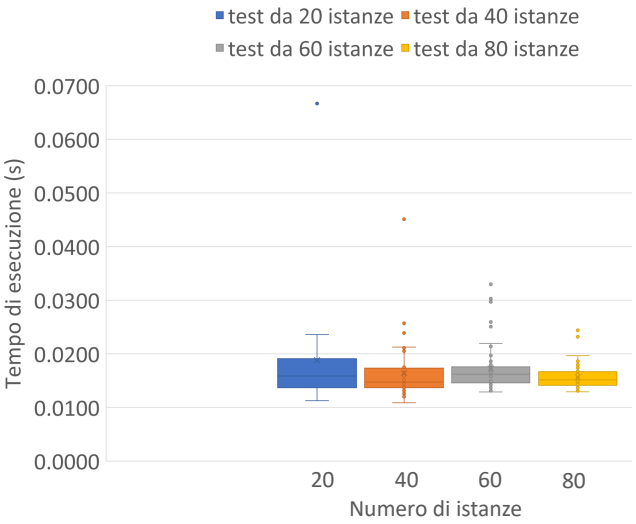
Bisogna puntualizzare che le misurazioni dei tempi di esecuzione comprendono, per ogni modulo, il tempo per creare e distruggere una connessione, il tempo per esplicitare la richiesta, il tempo di esecuzione del modulo che elabora la richiesta ed il tempo impiegato per la stampa dei feedback nel file di log:

$$t_{tot} = t_{connessione} + t_{richiesta} + t_{modulo} + t_{feedback}$$

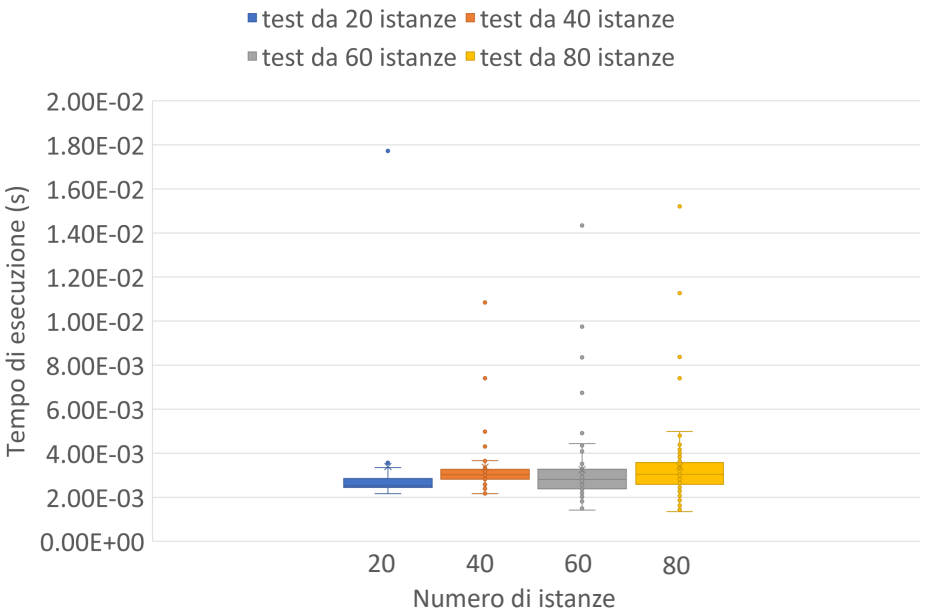
In Figura 8.2 viene rappresentata la distribuzione dei dati ottenuti dalla misurazione di performance del modulo di istanziazione della topologia, composto dalle funzioni di elaborazione e storage degli intenti, e del modulo di traduzione delle



(a) Modulo topologia virtuale NEMO-Python



(b) Modulo traduzione NEMO-Python



(c) Modulo traduzione NEMO-Java

Figura 8.2: Grafico scatola e baffi relativo al tempo di esecuzione dei moduli NEMO.

configurazioni, implementato exnovo. Dai grafici, presenti in figura, si osserva una distribuzione omogenea delle misurazioni, ovvero una bassa variabilità dei dati, dal momento che sia i baffi, che la differenza interquartile risultano ravvicinati tra loro, in tutti gli scenari.
All'interno dei tre grafici, tuttavia, sono presenti scale temporali differenti, in quanto il test in Figura 8.2c è svolto in contemporanea agli altri due, ma valuta il tempo effettivo di esecuzione del modulo al'interno del NEMO Engine, di fatto eliminando tutti i ritardi dovuti alla gestione della connessione con la piattaforma.

Premesso che tutti i test sono stati effettuati sempre nello stesso ambiente operativo e nelle medesime condizioni, la leggera variabilità che si può riscontrare è dovuta alla gestione del carico operativo da parte del sistema MGMT e del sistema

virtualizzato, tuttavia il degrado prestazionale risulta di un quantitativo che si può ritenere accettabile. Nella Tabella 8.1 contenete le mediane, ovvero il valore rappresentativo relativo dell'insieme di valori analizzati per ogni test di ogni modulo e non influenzabile da outliner[3], si può osservare come la variazione di performance maggiore, riferita alle somme dei tempi di esecuzione dei moduli analizzati, sia al massimo di $1.5e^{-3}$ secondi. Considerando che globalmente i tempi per l'istanziazione e creazione di un singolo file di configurazione si aggirano su valori al di sotto del decimo di secondo, considerando anche gli anni della piattaforma di test utilizzata, possiamo definire quanto osservato un ottimo risultato.

Test	Mediana modulo Topologia	Mediana modulo Traduzione	Somma
20 istanze	0.0142s	0.0159s	0.0301s
40 istanze	0.0140s	0.0147s	0.0287s
60 istanze	0.0135s	0.0162s	0.0297s
80 istanze	0.0151s	0.0151s	0.0302s

Tabella 8.1: Relazione tra valori rappresentativi dei test.

Come ultima analisi dei moduli riferiti alla piattaforma NEMO, si può osservare dalla Figura 8.3 come la regolarità nell'esecuzione delle istanze in tutti i test porta ad un'esecuzione predicibile di quest'ultimi, in quanto la crescita lineare delle richieste di file di configurazione fa crescere, al medesimo passo, anche il tempo di esecuzione. NEMO, quindi, risulta essere un'ottima componente sia per quanto riguarda l'affidabilità, sia per le prestazioni che riesce ad offrire.

8.2.2 Scenario operativo completo

Capita l'efficienza delle modifiche e dell'implementazione della piattaforma NEMO, occorre valutare le prestazioni della soluzione in uno scenario più ampio, di fatto andremo a considerare un ciclo completo di istanziazione di una configurazione.

Per testare lo scenario completo di configurazione si sono realizzati tre test differenti basati sulle funzionalità che sono previste dal sistema, rispettivamente la creazione di un servizio di sicurezza, la modifica di un servizio di sicurezza attivo

[3]Terminologia utilizzata in statistica per definire, in un insieme di osservazioni, un valore anomalo, chiaramente distante dalle osservazioni effettuate.

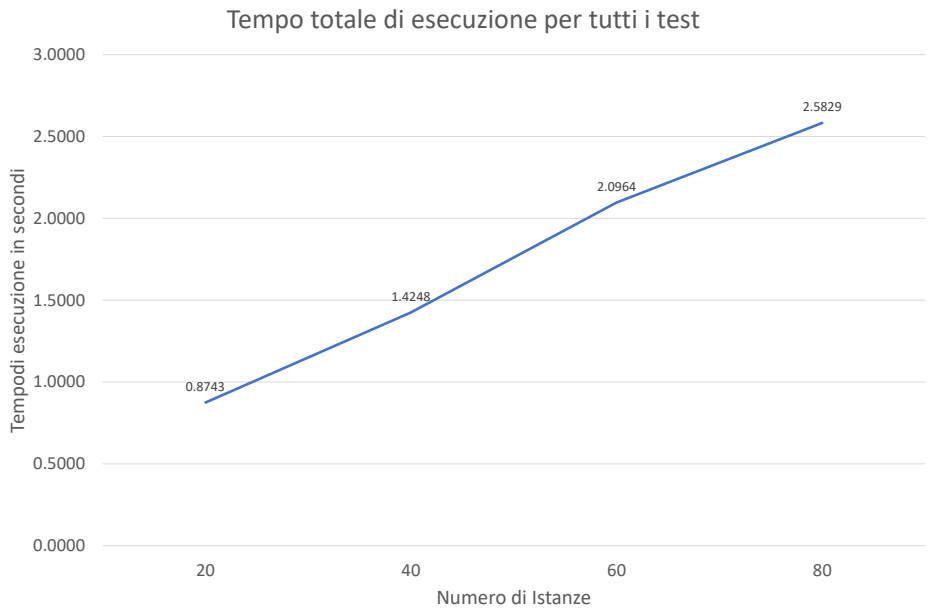


Figura 8.3: Grafico relativo al tempo di esecuzione dei test effettuati su NEMO.

e la cancellazione di un servizio di sicurezza esistente. Per ognuno dei tre test si è preso in considerazione l'apporto prestazionale di ogni singolo modulo, valutandoli dal lato dell'Orchestratore.

Il primo test, che prevede l'istanziazione di un WAF, è realizzato a partire da uno scenario privo di ogni tipo di informazione preliminare, sono attivi il modulo NEMO, l'Orchestratore e Kubernetes, ognuno nel proprio ambiente virtuale. Il test parte con l'attivazione dell'Orchestratore, il quale invia il seguente intento al modulo NEMO:

```
CREATE Node genericAppWAF Type waf Property application_to_protect:
    generic, alert_level: higth, service_port: 80,
    active_protection_attack: xss-sql-dos-php-java-rce-ip-shell,
    block_bad_user_agent: bot-scanner, block_country_traffic: es;
```

NEMO salva l'entità ricevuta ed elabora un file di configurazione in base alle caratteristiche specificate nel Data Store, a questo punto l'Orchestratore traduce il file di configurazione consegnatogli e, tramite una connessione SSH, implementa il WAF con le opportune caratteristiche all'interno di Kubernetes. Naturalmente ogni modulo invia aggiornamenti in tempo reale all'utente in base alle operazioni che esegue.

Il secondo test prevede l'aggiornamento del WAF, creato nel precedente scenario, a partire dal seguente intento:

```
UPDATE Node genericAppWAF Type waf Property application_to_protect:
    generic, alert_level: low, service_port: 80,
    active_protection_attack: xss-sql-dos-php-java-ip,
    block_bad_user_agent: bot, block_country_traffic: fr;
```

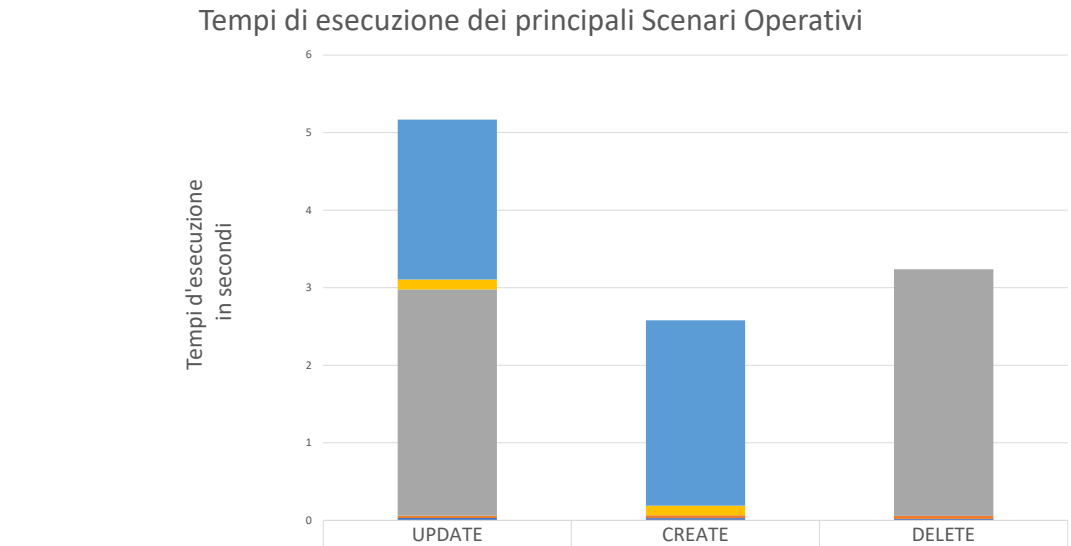
```
```

a differenza del primo test, qui l'esecuzione varia solo nella configurazione della piattaforma NFV, infatti, per garantire una pulizia del sistema target, vengono dapprima cancellate le configurazioni esistenti, riferite al WAF specificato, e poi ricaricate riavviando il servizio di sicurezza.

Il terzo ed ultimo test prevede la cancellazione del WAF istanziato in precedenza, sia all'interno della piattaforma NEMO che all'interno di Kubernetes, utilizzando l'intento:

```
1 DELETE Node genericAppWAF;
```

questo scenario possiede delle esecuzioni comuni al secondo test, in quanto entrambi devono cancellare le configurazioni già presenti all'interno della piattaforma NFV, con la differenza che, qui, il WAF viene cancellato e non riavviato.



	UPDATE	CREATE	DELETE
Modulo SSH Kubernetes	2.0623	2.3918	0.0000
Modulo traduzione Orchestratore	0.1285	0.1234	0.0000
cancellazione	2.9183	0.0000	3.1821
Modulo traduzione NEMO	0.0286	0.0366	0.0388
Modulo Topologia virtuale	0.0303	0.0281	0.0170

Figura 8.4: Grafico prestazionale relativo ai test operativi svolti.

In Figura 8.4 possiamo osservare il risultato prestazionale di tutti i test, paragonando i tempi di esecuzione di ogni singolo modulo.
Dalla Figura si evince che i tempi di utilizzo della piattaforma NEMO sono in linea con i risultati del paragrafo precedente, che il modulo di traduzione a basso livello

presente all'interno dell'Orchestratore è anch'esso abbastanza performante, paragonabile sia nel primo che nel secondo test, e che la comunicazione con la piattaforma NFV occupa più del 90% del tempo di esecuzione in tutti i test. L'eccessivo peso della comunicazione con Kubernetes è influenzato da vari fattori, ad esempio dal tempo di creazione della connessione, dal tempo di invio dei file di configurazione e dal tempo dovuto alle attese degli aggiornamenti che sono utilizzati come feedback verso l'utente.

Dai test risulta chiaro che la velocità per generare una configurazione di una funzione di sicurezza non dipenda dalla piattaforma NEMO, costante nelle sue operazioni, ma dalla piattaforma NFV che si utilizza e dalle configurazioni necessarie ad istanziare la vNSF.

Capitolo 9

Conclusioni

Lo studio intrapreso in questa tesi si è posto come obiettivo quello di valutare l'implementazione di un sistema per la semplificazione della gestione dei servizi di sicurezza, applicati alla rete, da parte dell'utente, in conformità delle richieste riportate nel progetto europeo FISHY. Ragion per cui, sono state condotte delle indagini per valutare gli strumenti più adatti al conseguimento degli obiettivi appena citati.
Tali investigazioni hanno condotto all'utilizzo di un approccio innovativo basato sui concetti di Intent-Based Networking e vNSF, adottando così il linguaggio IB-NEMO, grazie alla piattaforma realizzata da OpenDaylight, e la piattaforma Kubernetes, in linea con lo standard NFV proposto da ETSI.
Compreso lo scheletro di base per la costruzione del sistema, la ricerca si è concentrata sulla realizzazione di un WAF, studiando numerosi approcci software e metodi di configurazione. Questa ricerca ha selezionato il software ModSecurity come il più funzionale, in quanto consente grande flessibilità, alte prestazioni e dogmi di implementazione in linea con la filosofia del progetto.

L'implementazione del sistema proposto evidenzia la possibilità di implementare tale soluzione in un contesto applicativo reale, pertanto per valutare le scelte progettuali ed operative adottate si sono costruiti dei test, in grado di verificare le performance dell'intera soluzione.
I test, capaci di analizzare tutti i moduli del prototipo, hanno evidenziato dei risultati in linea con le aspettative della tesi, per quanto riguarda l'usabilità, la flessibilità e le performance del sistema. Le varie simulazioni effettuate, comprese quelle sugli scenari operativi completi, mettono in evidenza come l'implementazione di WAF, all'interno della piattaforma NFV utilizzata, impieghi circa 2,5791s per essere istanziata, di cui solo 0,0647s per interaggire con la piattaforma NEMO.
NEMO, quindi, si rivela uno strumento molto efficiente ed affidabile per gestire grandi moli di richieste, effettuate tramite intenti. Dall'analisi, tuttavia, si evince che la parte di esecuzione utilizzata per inviare feedback verso l'utente, implementata in tutti i moduli, non può essere trascurabile dal momento che consuma molte risorse in termini di gestione dell'IO; ancora, possiamo osservare come le performance dell'intero sistema siano profondamente correlate alla piattaforma NFV utilizzata e dalle configurazioni necessarie ad istanziare la vNSF.

Dai test effettuati e in previsione di uno scenario implementativo più ampio, si possono realizzare, sul prototipo, dei miglioramenti sia in termini di performance, sia in termini di usabilità. Uno dei primi interventi che si possono effettuare, in riferimento a tutte le componenti della soluzione, riguarda la possibilità di raccogliere automaticamente informazioni di rete, in modo da consentire a NEMO una visione dell'intera topologia su cui opera, senza ricorrere a configurazioni preliminari da parte dell'utente. Ancora, potrebbe essere interessante snellire il sistema di interazione NEMO-Orchestratore, accumulando gli intenti per poi eseguirli in parallelo, velocizzando sia le prestazioni che il numero di informazioni inviate come feedback che l'utente deve tenere sotto controllo.
In ultima analisi, dal punto di vista della sicurezza risulta sicuramente necessario spostare la comunicazione, dei metodi offerti dall'interfaccia di NEMO, su un protocollo più sicuro, come il TLS. Grazie all'utilizzo, da parte di NEMO, del web server Karaf, operando delle modifiche direttamente sui suoi file di configurazione, possiamo spostare la comunicazione dell'intera piattaforma dall'utilizzo del protocollo HTTP all'utilizzo del protocollo HTTPS.

Appendice A

Manuale Utente

L'appendice A definisce le linee guida per l'installazione e l'utilizzo del sistema prodotto durante lo svolgimento della tesi.
Il manuale parte con l'analisi dei requisiti necessari all'istallazione delle componenti e, a seguire, sarà illustrato come installare e configurare le componenti stesse, concludendo con una panoramica sull'utilizzo generale del sistema installato.
Si sottolinea che il manuale utente esamina l'istallazione e la configurazione delle componenti, che costituiscono il sistema, considerando l'utilizzo una piattaforma di MGMT basata su Linux-Debian o su Windows, per facilitare la riproducibilità del lavoro svolto.

A.1 Prerequisiti

Per l'istallazione e la compilazione di OpenDaylight NEMO e Kubernetes risultano nocessarie alcune operazioni preliminari, quali la configurazione di software predefiniti; si richiede, quindi, l'utilizzo di:

- Sistema MGMT: Ubuntu 20.04 LTS (Focal) oppure Windows 10 pro;
- Orchestratore di Container: Docker 20.10 LTS;
- Interprete: Python v3.6 o superiore;
- Compilatore: Java Development Kit;
- Server e client per la configurazione SSH: OpenSSH LTS;
- Sistema di virtualizzazione: Oracle VM VirtualBox 6.0 LTS.

Di seguito si analizzeranno le operazioni preliminari per istanziare i software propedeutici richiesti, tralasciando quelle per installare e configurare il sistema operativo della piattaforma di MGMT e quello per la macchina virtuale contenente NEMO ed Orchestratore.

A.1.1 Sistema di virtualizzazione

VirtualBox, ovvero il sistema di virtualizzazione, è fondamentale per per simulare gli attori principali presenti nella soluzione, introdotti nel Capitolo 6.

Per installare questo software, sia per sistemi Linux-Debian che per sistema Windows, occorre visitare il sito ufficiale[1], scaricare l'eseguibile adatto al propio sistema operativo e procedere con l'installazione.

A.1.2 Orchestratore di Container

L'istallazione della piattaforma di orchestrazione Docker risulta necessaria per il corretto funzionamento di Kubernetes, utilizzato per istanziare ModSecurity come vNSF; quindi, procediamo con l'installazione di questa componente sul sitema operativo di MGMT.

Per quanto riguarda l'istallazione di Docker su Windows 10, occore scaricare l'eseguibile di Docker-Desktop dalla pagina ufficiale[2] e procedere con l'istallazione.

Per quanto riguarda l'istallazione di Docker su sistemi Linux-Debian, occorre aprire il terminale ed utilizzare i seguenti comandi:

```
' i comandi di aggiornamento del sistema possono essere effettuati
una volta sola, durante la configurazione.'
$ sudo apt-get upgrade
$ sudo apt-get update
' ____________________________________________________ '
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Quando l'istallazione del software sarà completata si consiglia un riavvio del sistema operativo per consolidare le modifiche.

A.1.3 Interprete Python

L'istallazione dell'interprete Python è propedeutico al corretto funzionamento della piattaforma NEMO e all'esecuzione dell'Orchestratore. Oltre all'istallazione dell'interprete occorre scaricare ed installare tutte le librerie utilizzate per portare a termine le operazioni.
Nello scenario proposto dalla tesi, l'interprete viene installato all'interno del sistema operativo Ubuntu 20.04 LTS nella macchina virtuale contenente sia NEMO che l'Orchestratore.

Per effettuare la corretta istallazione occorre aprire il terminale e digitare i seguenti comandi:

[1] https://www.virtualbox.org/wiki/Downloads

[2] https://docs.docker.com/desktop/install/windows-install/

```
' ---------- OPZIONALI ----------
$ sudo apt-get upgrade
$ sudo apt-get update
 ------------------------------ '
$ sudo apt install python3-pip #installazione di python e pip
$ pip install paramiko
$ pip install -U Jinja2
$ pip install requests
$ pip install rstr
```

A.1.4 Compilatore Java

Per permettere la compilazione della piattaforma OpenDaylight NEMO occorre installare il Java Development Kit (JDK) versione 8, o superiore; invece, per automatizzare la costruzione delle dipendenze del software occorre installare Apache Maven[3], ampiamente sfruttato dalla piattaforma. Come per l'interprete Python, il JDK e l'Apache Maven vengono installati all'interno della macchina virtuale contenente Ubuntu 20.04 LTS, ragion per cui andiamo ad osservare i comandi necessari alla loro istanziazione all'interno di questo sistema operativo:

```
' ---------- OPZIONALI ----------
$ sudo apt-get upgrade
$ sudo apt-get update
 ------------------------------ '
#prima di tutto si procede con l'istallazione della JDK
$ sudo apt-get -y install openjdk-8-jdk
$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
#una volta istallata la JDK si procede con l'installazione di Apache
    Maven qualora non sia presente all'interno del sistema
$ wget https://dlcdn.apache.org/maven/maven-3/3.8.4/binaries/apache-
    maven-3.8.4-bin.tar.gz
$ sudo mkdir /usr/local/apache-maven
$ sudo mv apache-maven-3.5.4-bin.tar.gz /usr/local/apache-maven
$ sudo tar -xzvf /usr/local/apache-maven/apache-maven-3.5.4-bin.tar.gz
    -C /usr/local/apache-maven/
#Dopo aver scaricato ed estratto i dati si procede con l'istallazione
    di maven 3.5.4
$ sudo update-alternatives --install /usr/bin/mvn mvn /usr/local/
    apache-maven/apache-maven-3.5.4/bin/mvn 1
$ sudo update-alternatives --config mvn
```

Dopo aver effettuato l'istallazione della JDK e di Apache Maven occorre esporre, tramite variabile d'ambiente, i servizi all'interno del sistema:

```
$ export M2_HOME=/usr/local/apache-maven/apache-maven-3.5.4
#impostazioni di memoria per JVM che esegue Maven
$ export MAVEN_OPTS="-Xms256m -Xmx512m"
```

[3]Strumento di gestione per progetti software basati su Java e build automation.

A.1.5 Client-Server SSH

OpenSSH è il software principale che permette lo scambio di configurazioni per le vNSF, in modo sicuro, tra Orchestratore e piattaforma NFV.

L'istallazione su sistema operativo Windows avviene accedendo alle impostazioni di sistema, selezionando la sezione Applicazioni, attivando il pannello "Funzionalità facoltative" ed installando *OpenSSH Client* e *OpenSSH Server*.

Per quanto riguarda l'istallazione su sistema operativo Linux-Debian occorre aprire il terminale e digitare:

```
' ------------------------ OPZIONALI ------------------------------
$ sudo apt-get upgrade
$ sudo apt-get update
 ------------------------------------------------------------------ '
$ sudo apt-get install openssh-server
#una volta istallato Open SSH occorre abilitare il servizio
$ sudo systemctl enable ssh --now
#dopo aver abilitato il servizio locale occorre far partire il demone
    in background
$ sudo systemctl start ssh
```

A.2 Istallazione delle componenti principali

Si procede all'istallazione delle componenti principali, proposte dalla soluzione di questa tesi, configurando rispettivamente la piattaforma NEMO e la piattaforma Kubernetes.
Naturalmente le istanze delle piattaforme possono essere eseguite solo dopo l'effettiva predisposizione dell'ambiente operativo, ovvero la creazione delle macchina virtuali con i relativi software propedeutici (Sezione A.1).

A.2.1 Piattaforma NEMO

L'istallazione e la configurazione della piattaforma NEMO si attua in pochi e semplici passaggi manuali, oppure tramite l'utilizzo dello script "install.sh" fornito in allegato.
Il primo passo, per l'istallazione manuale, consiste nella creazione della cartella che ospiterà le configurazioni utilizzate da NEMO in fase di compilazione; per fare ciò si apre il terminale inserendo i comandi:

```
$ cd /home/mami
$ mkdir .m2
```

Dopo la creazione della cartella, si crea il file di configurazione che verrà utilizzato all'atto della compilazione:

```
$ cd /home/mami/.m2
$ touch settings.xml
$ cp -n $M2_HOME/.m2/settings.xml{,.orig} ; \wget -q -O - https://raw.
    githubusercontent.com/opendaylight/odlparent/master/settings.xml >
    $M2_HOME/.m2/settings.xml
```

A questo punto si procede con il download della repository github contenente l'intera struttura del progetto:

```
$ cd /home/mami
$ git clone https://github.com/mami-project/nemo.git
```

Prima di effettuare la compilazione bisogna applicare le modifiche discusse nel Capitolo 7. Per applicare le modifiche occorre sostituire, all'interno della cartella contenente i file di NEMO, le directory *demo* e *nemo-impl* allegate al lavoro effettuato.

Terminate le fasi preliminari, si può procedere alla compilazione della piattaforma:

```
'Ci si sposta all'esterno della cartella contenente i file binari di
    NEMO e si procede con la compilazione attraverso il gestore Apache
    Maven'
$ cd /home/mami
$ mvn clean install -DskipTests
```

Per completare le operazioni di installazione occorre avviare la piattaforma NEMO, attraverso il Web server Karaf, per installare i moduli necessari al suo corretto funzionamento. Per effettuare tali operazioni si consiglia l'utilizzo del terminale *XTerm*, già presente all'interno del sistema operativo:

```
'Ci si sposta all'interno della cartella contenente i file eseguibili
    di Karaf e si procede avviando il servizio '
$ cd /home/mami/nemo/nemo-karaf/target/assembly/bin
$ ./karaf
```

Una volta che il servizio è stato avviato con successo, come riportato nella Figura A.1, occorre installare i moduli precedentemente citati, attraverso il comando:

```
$ feature:install odl-nemo-api odl-nemo-engine odl-nemo-engine-rest
    odl-mdsal-all odl-restconf-all
```

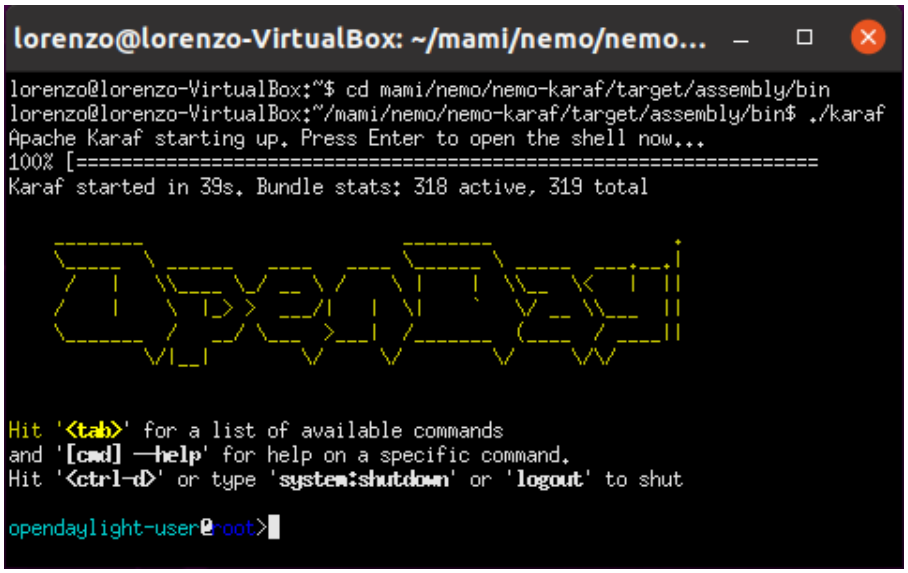


Figura A.1: Console XTerm rappresentante NEMO pronto alla configurazione

Terminata l'istallazione di tutti i moduli, la piattaforma NEMO è pronta a ricevere ed elaborare intenti. Per qualunque approfondimento e analisi del progetto si può fare riferimenti alla repository github[4] utilizzata.

A.2.2 Piattaforma Kubernetes

La piattaforma Kubernetes, come ampiamente spiegato nel Capitolo 7, viene istanziata come macchina virtuale *Minikube* all'interno del software VirtualBox installato all'interno del sistema operativo Windows 10 pro. In questa sottosezione si mostra come effettuare l'installazione e la configurazione di questa piattaforma sia sul sistema operativo proposto dalla soluzione, sia su sistemi Linux-Debian.

Per quanto riguarda l'istallazione di Minikube all'interno Windows 10pro occorre accedere alla pagina ufficiale del sistema[5] scaricare l'eseguibile *minikube-installer.exe* e procedere con l'istallazione automatica.

Per quanto riguarda i sistemi Linux-Debian occorre aprire il terminale e lanciare i seguenti comandi:

```
1 'Ci si sposta all'esterno della cartella contenente i file binari di
    NEMO e si procede con la compilazione attraverso il gestore Apache
    Maven'
2 $ ccurl -LO https://storage.googleapis.com/minikube/releases/latest/
    minikube-linux-amd64
3 $ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Bisogna specificare che, per le ragioni indicare nel Capitolo 7, non occorre modificare le impostazioni proposte di default dall'eseguibile in quanto si utilizzeranno

[4] https://github.com/mami-project/nemo

[5] https://minikube.sigs.k8s.io/docs/start/

i requisiti minimi richiesti per l'avvio della macchina virtuale.
Alla conclusione delle operazioni, in entrambi i sistemi operativi, è consigliabile un riavvio delle console di comando aperte per consolidare le modifiche.

Per avviare la macchina virtuale Minikube istanziata nel sistema MGMT, bisogna inserire, nel caso di Windows 10 pro tramite il terminale powershell avviato come amministratore, nel caso di sistemi Linux-Debian tramite il terminale di sistema, il seguente comando:

```
> minikube start
```

Di seguito si riportano una serie di comandi utili all'interazione con Minikube:

```
#comando che mostra tutti i servizi attivi ed esposti all'esterno
    tramite docker container
> minikube service list

#comando utilizzato per arrestare la macchina virtuale
> minikube stop
```

A.3 Utilizzo della soluzione

In questa sezione si espone come replicare l'istanziazione di un WAF tramite la sottomissione di un intento a NEMO. L'obiettivo è quello di comprendere il sistema in tutte le sue componenti, attraverso lo studio di un esempio concreto pertinente ai concetti esposti nella tesi.

A.3.1 Utilizzo di NEMO

Al termine dell'istallazione di NEMO e avviato il servizio Karaf, si può interagire correttamente con la piattaforma. I passi da seguire per permettere una corretta interazione riguardano:

- configurazione delle entità previste da NEMO, introdotte nella Sezione 4.2.1;
- creazione di un utente amministratore, per consentire la comunicazione;
- creazione di un file contenente gli intenti da elaborare.

Per definire le entità Nodo, Connessione e Flusso si può utilizzare il modulo di configurazione presente nella cartella */home/mami/nemo/demo*, inserendo nel terminale della macchina virtuale, in cui è istanziata la piattaforma, il seguente comando:

```
'Ci si sposta all'interno della cartella demo e si procede con la
    configurazione della piattaforma'
$ cd /home/mami/nemo/demo
$ python3 -d ./config.py
```

Terminate le operazioni per l'inserimento delle entità, si procede con la creazione di un file contenente gli intenti da elaborare e con la creazione di un account amministratore.
Per semplificare tali operazioni sono stati inseriti all'interno della cartella demo diversi file, tra cui *intent.txt*, contenente la costruzione di un WAF tramite intento, il file *orchestratore.py*, in grado di interagire sia con la piattaforma NEMO che con la piattaforma Kubernetes, e infine il file *readme.txt*, in cui vengono riportate delle informazioni generarli riguardanti l'utilizzo di tutti i file presenti.

Grazie al modulo software *orchestratore.py* possiamo, tramite dichiarazioni esplicite, effettuare la creazione di un account amministratore, prelevare gli intenti contenuti in un file e consentire operazioni di comunicazione verso la piattaforma NEMO, finalizzate all'elaborazione di intenti. Naturalmente *orchestratore.py* consente, anche, di istanziare concretamente le entità virtuali presenti all'interno di NEMO sulla piattaforma Kubernetes.
Per le ragioni sopra esposte è possibile avviare un corretto flusso di operazioni, per interagire con la piattaforma NEMO, inserendo nel terminale il comando:

```
$ cd /home/mami/nemo/demo
$ python3 -d ./orchestratore.py --intent intent.txt --instance Nan --
    style generic
```

Il comando sopra esposto consente di registrare un account amministratore all'interno della piattaforma, effettuare il login e avviare una transazione con l'obiettivo di elaborare gli intenti contenuti nel file *intent.txt*. Qualora l'account amministratore sia già stato creato all'interno della piattaforma, il modulo salterà la fase preliminare di registrazione effettuando direttamente login e comunicazione degli intenti da elaborare.

Quanto riportato in questa sottosezione consente, ad un utente, di familiarizzare con la piattaforma NEMO, e con l'omonimo linguaggio basato su intenti, attraverso un'esecuzione esemplificativa guidata.
Naturalmente, si approfondiranno possibili modifiche alle configurazioni preliminari presentate, contenute nel modulo software *config.py*, *orchestratore.py* e *intent.txt*, all'interno dell'Appendice B.

A.3.2 Utilizzo dello Scenario Operativo

In Figura A.2 viene riportato lo schema dell'ambiente operativo, istanziato nelle Sezioni precedenti, utilizzato per replicare le operazioni condotte durante i test del Sottoparagrafo 8.2.2.

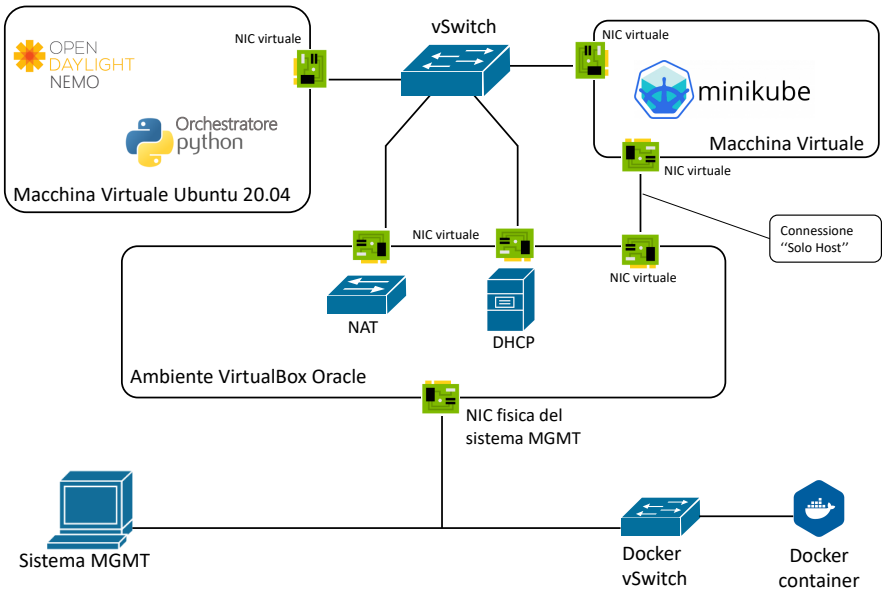


Figura A.2: Configurazione dell'ambiente operativo

In particolare, in questa Sottosezione si vuole permettere la comprensione del funzionamento generale del prototipo, proposto dalla tesi, ad un generico utente.

Per consentire l'istanziazione completa del WAF ModSecurity occorre avviare, oltre alla macchina virtuale Ubuntu 20.04, la macchina virtuale Minikube e i servizi di comunicazione OpenSSH. Quando l'ambiente operativo risulta essere pronto, con NEMO configurato (fare riferimento alla Sezione A.3.1) e i servizi di comunicazione attivi, bisogna effettuare delle operazioni per consentire la corretta configurazione del WAF. In particolare si rende necessario creare una cartella locale al sistema MGMT, che sarà utilizzata come volume virtuale da Minikube; all'interno di questa cartella dovranno essere inseriti i file, forniti in allegato alla tesi, *GeoIP.dat*, database utilizzato per identificare l'area geografica dell'ip che effettua le richieste, e *nginx.conf*, utilizzato per configurare l'immagine docker contenente ModSecurity. Dopo aver inserito la cartella nel sistema MGMT occorre modificare, all'interno della macchina virtuale Ubuntu 20.04, il template *k8s-waf.txt*, contenuto nel percorso */home/mami/nemo/demo/templates*, sostituendo i path contenuti nella sezione *volumes* con il path della cartella appena creata:

```
'si riporta l'estratto del file che bisogna modificare'
volumes:
    # You set volumes at the Pod level, then mount them into
    containers inside that Pod
    - name: test
      hostPath:
        path: "/path_della_cartella_creata"
        type: DirectoryOrCreate
    - name: nginx
      hostPath:
        path: "/path_della_cartella_creata/nginx.conf"
        type: FileOrCreate
```

si consiglia di scrivere il path nel formato */directory principale/sottodirectory*, sia per sistemi operativi Windows che Linux-Debian.
Terminate le operazioni per consentire la configurazione di ModSecurity, si passa alle operazioni per consentire una corretta comunicazione tra Orchestratore e Minikube; a tal fine, si rende necessario inserire all'interno del file *orchestratore.py* le credeziali del sistema di MGMT, in modo da poter effettuare la comunicazione SSH tra le componenti. Per concludere la configurazione del modulo Orchestratore occorre modificare, nella sezione del comunicazione con Kubernetes, il path di destinazione dei file di configurazione del sistema MGMT.

Per eseguire, quindi, lo scenario completo occorre inserire nel terminale della macchina virtuale Ubuntu 20.04 i seguenti comandi:

```
$ cd /home/mami/nemo/demo
$ python3 -d ./orchestratore.py --intent intent.txt --instance
    genericWAF --style generic
```

Il modulo che permette di avviare lo scenario completo, tramite il comando appena definito, riceve in ingresso il file contenente l'intento *intent.txt*, il nome dell'entità da istanziare, che nel caso in esame coincide con il nome del WAF rappresentato nel file di intenti, e la tipologia di traduzione che NEMO deve adottare.
Si deve specificare che il modulo fornisce, per tutta la durata delle operazioni, aggiornamenti all'utente in modo da scandirne la corretta esecuzione.
Alla fine delle operazioni sarà possibile controllare lo stato dei servizi attivi all'interno di Minikube tramite i comandi introdotti nella Sezione A.2.2.

Per agevolare l'utilizzo dello scenario operativo completo si riportano nella Tabella A.1 le proprietà ed i valori accettati dagli intenti finalizzati alla costruzione di un WAF tramite la piattaforma NEMO.

Nome Proprietà	Valori accettati	Descrizione
active_protection_attack	xss, sql, dos, php, java, rce, ip, shell	Questa proprietà consente di specificare le tipologie di attacco che il WAF deve essere in grado di fermare. Tali valori possono essere specificati contemporaneamente all'interno dell'attributo, separati dal carattere "-".

Nome Proprietà	Valori accettati	Descrizione
block_bad_user_agent	bot, scanner, crawler	Questa proprietà consente di specificare la volontà di voler bloccare i principali metodi per scannerizzare un'applicazione web. Come per la proprietà active_protection_attack, si possono specificare più valori contemporaneamente.
block_country_traffic	it, es, fr etc	Questa proprietà permette al WAF di accettare solo il traffico proveniente da una particolare area geografica, si può specificare una sola area geografica alla volta, tramite la nomenclatura standard dell'*ISO 3166-2*.
alert_level	none, low, medium, higth, extreme	Questa proprietà serve per capire quanto restrittive devono essere le regole di configurazione del WAF. Più il livello di allerta è alto, maggiore sarà la schermatura del WAF; naturalmente al crescere del livello di allerta si rischia di bloccare anche comunicazioni sicure, scambiandole per dannose.
block_domain_name	FQDN valido	Questa proprietà consende di bloccare connessioni da e verso un particolare FQDN.

Tabella A.1: Proprietà dell'intento per la crazione di WAF.

Appendice B

Manuale Sviluppatore

L'appendice B ha lo scopo di descrivere i componenti sviluppati in questa soluzione, nell'ottica di permettere estensioni che divergono dallo scenario proposto.

B.1 NEMO

B.1.1 Istallazione packages

Grazie all'utilizzo di *Apache Karaf*, è possibile estendere le funzionalità di NEMO attraverso l'istallazione di pacchetti applicativi aggiuntivi. In questa Sottosezione, quindi, si analizzerà la documentazione[1] di karaf per permettere un'estensione semplice e rapida delle funzionalità di NEMO attraverso l'istallazione di packages dedicati.

Per permettere l'istallazione di nuove funzionalità, occorre avviare karaf (vedi Sezione A.3.1) e digitare un comando che rispetti la seguente sintassi generica:

```
1 $ features:<comando> [<opzioni>+] [<parametro>]
```

Nello specifico possiamo utilizzare uno dei comandi di ispezione dei packages messi a disposizione, che si compone come segue:

```
1 $ features:list [<opzioni>]
2 # -o visualizza i pacchetti in ordine alfabetico
3 # -i visualizza i pacchetti attualmente installati
4 $ features:info [<opzioni>] nome
5 # -c mostra informazioni sulla configurazione
6 # -d mostra informazioni sulle dipendenze
```

Selezionato il pacchetto di interesse, ovvero quello che si vuole installare, è possibile attivarlo attraverso il seguente comando:

[1] https://karaf.apache.org/manual/latest-2.x/commands/commands.html

```
$ features:install [<opzioni>] nome
$ features:uninstall [<opzioni>] nome # per disattivare i pacchetti
    attivi
```

Un esempio di package, che è possibile istanziare per estendere l'usabilità di NEMO, è quello in grado di fornire un'interfaccia grafica capace di permette un utilizzo ad alto livello della piattaforma, nonchè la visualizzazione delle strutture dati presenti al suo interno. In particolare, per utilizzare tale funzionalità, occorre installare i pacchetti *odl-dluxapps-topology* e *odl-dluxapps-yangui* capaci di fornire una visualizzazione della topologia interna alla piattaforma NEMO, similmente ad un controllore di SDN. Nella Figura B.1 è possibile vedere il risultato del modulo *odl-dlux* ottenuto, previa autenticazione, collegandosi al servizio, tramite l'URL `http://{IP-locale}:8181/dlux/index.html#/topology/index`.
Un'altro modulo che vale la pena citare è *odl-nemo-engine-ui* capace di fornire supporto grafico all'utilizzo degli strumenti offerti da NEMO.

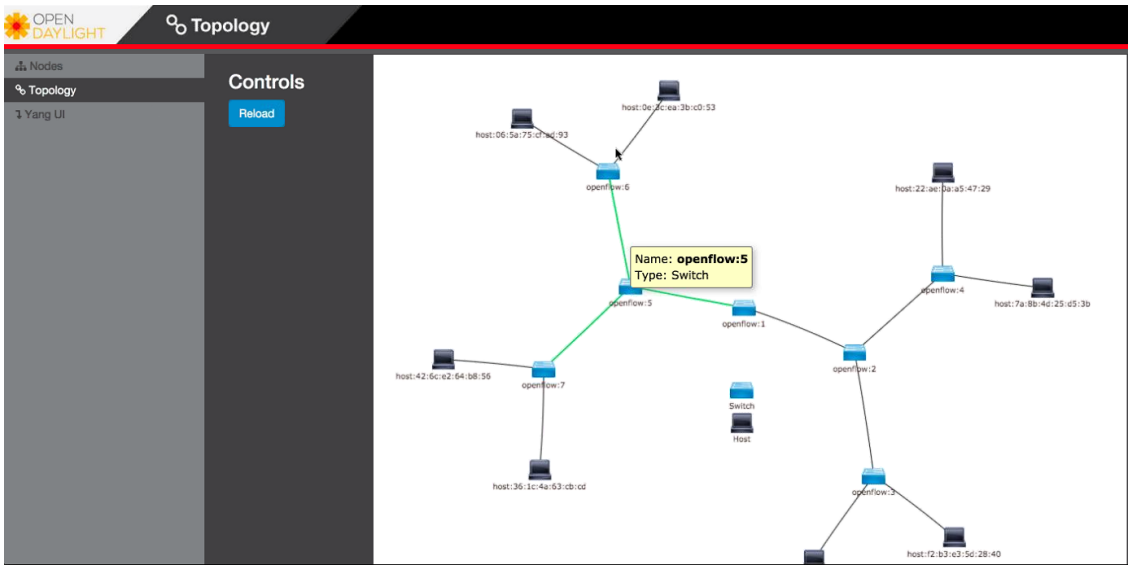


Figura B.1: Esempio di topologia attraverso il modulo Dlux[2]

B.2 Orchestratore

In questa Sezione si analizzeranno le componenti principali dell'orchestratore, argomentando in particolare le caratteristiche delle funzioni da cui esso è composto.

Per effettuare una prima configurazione delle entità di NEMO, come riportato nell'Appendice A, occorre utilizzare le informazioni contenute nel file *config.py*;

[2] https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/_viewing_network_topology.html

all'interno di questo modulo sono presenti numerose chiamate di funzioni che permettono la configurazione preliminare delle entità principali supportate da NEMO. In particolare vengono delineate le entità Nodo, Connessione e Flusso, con le rispettive proprietà, ancora vengono definiti i ruoli accettati per un nuovo utente e molte altre strutture dati.
Tali informazioni vengono inviate sotto forma di file di configurazione JSON, estendibile liberamente a patto di rispettare la sintassi dell'entità che si vuole modificare o aggiungere. Durante l'invio delle configurazioni è possibile ricevere degli aggiornamenti visivi, che scandiscono la corretta esecuzione delle operazioni.

All'interno del file *orchestratore.py* vengono riportate le funzioni che permettono l'esecuzione completa dell'istanza di un WAF; in particolare il file viene suddiviso in moduli ben distinti.
Nel caso in esame troviamo un set di funzioni capaci di interfacciarsi con NEMO, al fine di sottomettere una serie di intenti che devono essere elaborati. Tali funzioni eseguono una serie di operazioni di tipo POST che permettono la registrazione di utenti, con relativi ruoli, di inizializzare transazioni, di sottomettere intenti e di generare file di configurazioni completi per le entità salavate all'interno della piattaforma. Queste funzioni non necessitano di modifiche, in quanto le operazioni di interazione risultano ben definite.
Per quanto riguarda il modulo di traduzione al momento è in grado di valutare scenari generati da NEMO solo per quanto riguarda le vNSF di tipo WAF; in particolare genera file di configurazione ModSecurity finalizzati alla modifica di un container Docker. Risulta possibile integrare nuove vNSF ampliando tale modulo con nuove sezioni relative a differenti vNSF.
In ultima analisi si riscontra il modulo di interazione con la piattaforma Kubernetes, il quale tramite sessione SSH, invia comandi alla shell del sistema Windows 10 pro al fine di instanziare nuovi servizi all'interno di Minikube. Questo particolare modulo permette l'istanziazione generica di container docker all'interno della piattaforma Kubernetes, ma può essere rimosso e sostituito in toto, in base alla piattaforma NFV che si vuole utilizzare.

I file citati in questa Sottosezione possono essere facilmente compresi, in quanto ogni singola funzione viene corredata di commento.

B.3 Configurazioni di ModSecurity

Le configurazioni proposte per ModSecurity WAF corrispondono ad una serie di file preimpostati, inseriti nelle cartelle presenti in */home/nami/nemo/demo/mod_sec_conf*, suddivisi in base alla tipologia di attacco che possono bloccare. Tali file possono essere ampliati aggiungendo, rimuovendo o modificando le regole presenti al loro interno, rispettando il linguaggio SecRule introdotto nella Sezione 5.3.3.

Si evidenzia che la configurazione di ModSecurity viene generata per un container Docker che dovrà essere implementato all'interno della piattaforma Kubernetes; per queste ragioni, all'interno del percorso */home/nami/nemo/demo/templates*, viene implementato il file di configurazione del container tramite il template YAML *k8-waf.txt*. Il file *k8-waf.txt* viene elaborato dall'orchestratore, tramite la libreria

jinja, inserendo alcuni valori fondamentali per permetterne il corretto funzionamento all'interno della piattaforma NFV. Qualora si decida di sostituire il modulo di interazione con Kubernetes, sarà possibile cambiare il template di istanziazione del WAF concordemente alla piattaforma selezionata, mantenedno inalterata la parte di generazione delle configurazioni per il container docker.

B.4 Integrazione di nuove vNSF

La soluzione proposta da questa tesi può essere utilizzata per istanziare una moltitudine di funzioni di sicurezza, su topologie rete più o meno complesse. Per queste ragioni, risulta necessario fornire delle linee guida che permettano l'espansione delle vNSF istanziabili.

B.4.1 Estensione della sintassi

Il primo problema che si affronta, quando si vogliono espandere le funzionalità del prototipo attraverso l'aggiunta di vNSF, riguarda la corrispondenza tra il linguaggio intent-based utilizzato e le entità da realizzare; in particolare, si rende necessario espandere la sintassi in modo da integrare la vNSF all'interno della piattaforma NEMO. Per risolvere questa prima problematica occorre valutare se le operazioni fornite dalla piattaforma siano sufficienti, procedendo quindi alla modifica del file JSON, necessario alle configurazioni preliminari di NEMO, aggiungendo in modo opportuno le nuove entità che si vuole siano supportate, esattamente come riportato nel Capitolo 7.
Qualora le operazioni fornite da NEMO non risultino sufficienti, si rende necessario estendere le sue funzionalità.
Per estendere la sintassi di NEMO, quindi, occorrono molti interventi strutturali, i quali, per semplicità, saranno elencati ed illustrati in ordine:

1. la prima operazione da effettuare riguarda la modifica dei file YANG utilizzati da MD-SAL in fase di configurazione, per creare le classi delle entità di base utilizzate da NEMO.
   I file YANG si trovano all'interno del percorso */home/mami/nemo/nemo-api/src/main/yang*;

2. ottenute le prime modifiche riguardanti la composizione delle entità di base, occorre modificare il parser per consentire la corretta interpretazione degli intenti; per fare ciò è necessario modificare il file *LanguageStyle.jj* presente in *nemo-impl/src/main/java/org/opendaylight /nemo/user/vnspacemanager/languagestyle/NEMOParse*.
   LanguageStyle.jj è il file che deve essere interpretato da JavaCC per creare il parser, ragion per cui ad ogni modifica occorre rigenerare le classi Java da sostituire a quelle obsolete, prima di effettuare nuovamente la compilazione della piattaforma;

3. dopo la modifica del parser per la corretta interpretazione degli intenti, occorre rimappare le modifiche previste dai file oggetto MD-SAL all'interno del

backend del NEMO-engine, per fare ciò occorre modificare le classi template contenute nel percorso *nemo-impl/src/main/java/org/opendaylight /nemo/user/vnspacemanager/structurestyle* in modo da consolidare le modifiche;

4. come ultima operazione, occorre testare la stabilità e la funzionalità del sistema NEMO, dopo le modifiche effettuate.

Allegato alla tesi viene fornito un esempio di modifica che rispetta le linee guida presentate in questa sottosezione.

B.4.2 Estensione del modulo di traduzione

Il secondo problema che si affronta, durante l'aggiunta di nuove vNSF nel sistema, coincide con la generazione dei file di configurazione prodotti da NEMO e successivamente tradotti dall'orchestratore. Per effettuare le modifiche all'interno della piattaforma NEMO occorre modificare il file *nemo-impl/src/main/java/org/ opendaylight/nemo/user/processingmanager/VNFDGenericManager.java*, ampliando il sistema di mappatura delle proprietà in base all'entità che si sta analizzando.

Per quanto riguarda la modifica del modulo di traduzione interno all'Orchestratore, per generare le configurazioni di basso livello, si rende necessario ampliare il file *orchestratore.py* per permettere una corretta traduzione in base alla piattaforma NFV adottata.

In conclusione, le modifiche che si possono applicare alla soluzione presentata in questa tesi possono essere molteplici, di entità e difficoltà variabile; per queste ragioni risulta fondamentale acquisire esperienza sul funzionamento del sistema, per mezzo dell'appendice A, ed effettuare numerosi test per capire i meccanismi che regolano queste componenti software.

Bibliografia

[1] E. Zeydan and Y. Turk, "Recent Advances in Intent-Based Networking: A Survey", 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), Conferenza virtuale, 25 Maggio - 31 Giugno, 2020, pp. 1–5, DOI 10.1109/VTC2020-Spring48590.2020.9128422

[2] E. Demaria, A. Pinnola, and N. Santinelli, "La virtualizzazione di rete: lo standard nfv", Telecom Italia, Febbraio 2015. https://www.gruppotim.it/content/dam/telecomitalia/it/archivio/documenti/Innovazione/MnisitoNotiziario/2015/2-2015/capitolo-07/articolo07.pdf, Ultimo accesso il 10-09-2022

[3] A. Desai, "The definitive guide to virtual platform management", Realtimepublishers. com, 2007, ISBN: 9781931491730

[4] R. Bolla, C. Lombardo, R. Bruschi, and S. Mangialardi, "Dropv2: energy efficiency through network function virtualization", IEEE Network, vol. 28, no. 2, 2014, pp. 26–32, DOI 10.1109/MNET.2014.6786610

[5] P. M. Mell and T. Grance, "The NIST definition of cloud computing. SP 800-145", National Institute of Standards and Technology, Settembre 2011, pp. 800–145. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf, Ultimo accesso il 10-09-2022

[6] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation", IEEE Communications Surveys & Tutorials, vol. 16, no. 4, Maggio 2014, pp. 2181–2206, DOI 10.1109/COMST.2014.2326417

[7] European Telecommunications Standards Institute, "ETSI GS NFV 002 V1.1.1 (2013-10) Network Functions Virtualisation (NFV); Architectural Framework." https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf, Ottobre 2013, Ultimo accesso il 10-09-2022

[8] European Telecommunications Standards Institute, "ETSI GS NFV-SWA 001 V1.1.1 (2014-12) Network Functions Virtualisation (NFV); Virtual Network Functions Architecture." https://www.etsi.org/deliver/etsi_gs/nfv-swa/001_099/001/01.01.01_60/gs_nfv-swa001v010101p.pdf, Dicembre 2014, Ultimo accesso il 10-09-2022

[9] European Telecommunications Standards Institute, "ETSI GR NFV 001 V1.2.1 (2017-05) Network Functions Virtualisation (NFV); Use Cases." https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf, Maggio 2017, Ultimo accesso il 10-09-2022

[10] European Telecommunications Standards Institute, "ETSI GS NFV-INF 001 V1.1.1 (2015-01) Network Functions Virtualisation (NFV); Infrastructure Overview." https://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_nfv-inf001v010101p.pdf, Gennaio 2015, Ultimo accesso il 10-09-2022

[11] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs Containerization to Support PaaS", 2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, 11-14 Marzo, 2014, pp. 610–614, DOI 10.1109/IC2E.2014.41

[12] S. Natarajan, R. Krishnan, A. Ghanwani, D. Krishnaswamy, P. Willis, and A. Chaudhary, "IETF - An Analysis of Container-based Platforms for NFV." https://www.ietf.org/proceedings/94/slides/slides-94-nfvrg-11.pdf, Gennaio 2015

[13] Docker Inc, "Documentazione ufficiale docker." https://docs.docker.com, Ultimo accesso il 10-09-2022

[14] European Telecommunications Standards Institute, "ETSI GS NFV-MAN 001 V1.1.1 (2014-12) Network Functions Virtualisation (NFV);Management and Orchestration." https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf, Dicembre 2014, Ultimo accesso il 10-09-2022

[15] Google llc, "Kubernetes project." https://github.com/kubernetes/kubernetes, Ultimo accesso il 10-09-2022

[16] Google llc, "Kubernetes documentation." https://kubernetes.io/it/docs, Ultimo accesso il 10-09-2022

[17] M. Siddiqui, T. Radi, L. Obuchowicz, and P. Rutkowski, "Enabling new features with kubernetes for nfv." https://builders.intel.com/docs/networkbuilders/enabling_new_features_in_kubernetes_for_NFV.pdf, 2017, Ultimo accesso il 10-09-2022

[18] J. Kephart and D. Chess, "The vision of autonomic computing", Computer, vol. 36, no. 1, Gennaio 2003, pp. 41–50, DOI 10.1109/MC.2003.1160055

[19] M. H. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. Carpenter, S. J. H. Campus), and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals (RFC-7575)." https://www.rfc-editor.org/rfc/rfc7575.html, Giugno 2015

[20] S. Jiang, B. Carpenter, and M. H. Behringer, "General Gap Analysis for Autonomic Networking (RFC-7576)." https://www.rfc-editor.org/rfc/rfc7576.html, Giugno 2015

[21] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Definitions." https://www.ietf.org/archive/id/draft-irtf-nmrg-ibn-concepts-definitions-09.html, Marzo 2022, Ultimo accesso il 10-09-2022

[22] C. Janz, N. Davis, D. Hood, M. Lemay, D. Lenrow, L. Fengkai, F. Schneider, J. Strassner, and A. Veitch, "Intent NBI – Definition and Principles (ONF TR-523)." https://opennetworking.org/wp-content/uploads/2014/10/TR-523_Intent_Definition_Principles.pdf, Ottobre 2016, Ultimo accesso il 10-09-2022

[23] A. Lerner, "Intent-based Networking." https://blogs.gartner.com/andrew-lerner/2017/02/07/intent-based-networking/, Febbraio 2019, Ultimo accesso il 10-09-2022

[24] "What is Intent-Based networking (IBN)?." https://www.vmware.com/topics/glossary/content/intent-based-networking.html, Ultimo accesso il 10-09-2022

[25] P.-J. Nefkens, "Intent-based Networking." https://www.ciscopress.com/articles/article.asp?p=2995353, Febbraio 2020, Ultimo accesso il 10-09-2022

[26] European Telecommunications Standards Institute, "ETSI GR ZSM 005 V1.1.1 (2020-05) Zero-touch network and Service Management (ZSM); Means of Automation." https://www.etsi.org/deliver/etsi_gr/ZSM/001_099/005/01.01.01_60/gr_zsm005v010101p.pdf, Maggio 2020

[27] M. H. Behringer, B. E. Carpenter, T. Eckert, L. Ciavaglia, and J. C. Nobre, "A Reference Model for Autonomic Networking." RFC 8993, May 2021, DOI 10.17487/RFC8993

[28] L. Pang, C. Yang, D. Chen, Y. Song, and M. Guizani, "A Survey on Intent-Driven Networks", IEEE Access, vol. 8, Gennaio 2020, pp. 22862–22873, DOI 10.1109/ACCESS.2020.2969208

[29] D. Borsatti, W. Cerroni, G. Davoli, and F. Callegati, "Intent-based Service Function Chaining on ETSI NFV Platforms", 2019 10th International Conference on Networks of the Future (NoF), Roma, Lazio, Italia, 1-3 Ottobre 2019, pp. 144–146, DOI 10.1109/NoF47743.2019.9015069

[30] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining Network Intents for Self-Driving Networks", Proceedings of the Afternoon Workshop on Self-Driving Networks, New York, NY, USA, 24 Agosto 2018, p. 15–21, DOI 10.1145/3229584.3229590

[31] A. S. Jacobs, R. J. Pfitscher, R. H. Ribeiro, R. A. Ferreira, L. Z. Granville, W. Willinger, and S. G. Rao, "Hey, Lumi! Using Natural Language for Intent-Based Network Management", 2021 USENIX Annual Technical Conference (USENIX ATC 21), Conferenza virtuale, July 14-16 Luglio 2021, pp. 625–639

[32] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A Language for Managing Network Resources", IEEE/ACM Transactions on Networking, vol. 26, no. 5, Settembre 2018, pp. 2188–2201, DOI 10.1109/TNET.2018.2867239

[33] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi, "Tierless Programming and Reasoning for Software-Defined Networks", 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), Seattle, WA, USA, 2-4 Aprile 2014. https://cs.brown.edu/~sk/Publications/Papers/Published/nfsk-flowlog-tierless/paper.pdf, Ultimo accesso il 10-09-2022

[34] S. Hares, "Intent-Based Nemo Overview", Internet-Draft draft-hares-ibnemo-overview-01, Internet Engineering Task Force, October 2015. https://datatracker.ietf.org/doc/draft-hares-ibnemo-overview/01/ ancora in sviluppo, ultimo accesso il 10-09-2022

[35] Huawei, "Intent NBI for software defined networking." http://www-file.huawei.com/~/media/CNBG/Downloads/Technical%20Topics/

Fixed%20Network/Intent%20NBI%20for%20Software%20Defined%20Networking-whitepaper, 2016

[36] Y. Xia, S. Jiang, T. Zhou, S. Hares, and Y. Zhang, "Nemo (network modeling) language." https://www.potaroo.net/ietf/idref/draft-xia-sdnrg-nemo-language, Ottobre 2015

[37] C. Cain and M. K. Barnwal, "Opendaylight nemo." https://wiki.opendaylight.org/display/ODL/NEMO, Agosto 2021

[38] OpenDaylight, "Opendaylight nemo: User manual." https://wiki-archive.opendaylight.org/view/NEMO:User_Manual, Ultimo accesso il 10-09-2022

[39] OpenDaylight, "Opendaylight nemo: Architetture." https://wiki-archive.opendaylight.org/view/NEMO:Architecture, Ultimo accesso il 10-09-2022

[40] K. Ingham, S. Forrest, *et al.*, "A history and survey of network Firewalls", University of New Mexico, Tech. Rep, Dicembre 2002

[41] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin, "Firewalls and Internet Security: Repelling the Wily Hacker", Addison-Wesley Longman Publishing Co., Inc., 2 ed., Febbraio 2003, ISBN: 020163466X

[42] W. Stallings, "Sicurezza delle reti: Applicazioni e standard", Paravia Bruno Mondadori Editori, 3 ed., Aprile 2007, ISBN: 9788871923451

[43] M. Ly, D. Taniar, and M. Abi-Raad, "Firewall: Technologies and Practices", World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI) 2001, Orlando, FL, USA, 22-25 Luglio 2001, pp. 575–580. https://users.monash.edu.au/~dtaniar/SCI2001-Mary.pdf

[44] D. J. D. Touch, "Updated Specification of the IPv4 ID Field." RFC 6864, Febbraio 2013, DOI 10.17487/RFC6864

[45] "Transmission Control Protocol." RFC 793, Settembre 1981, DOI 10.17487/RFC0793

[46] "User Datagram Protocol." RFC 768, Agosto 1980, DOI 10.17487/RFC0768

[47] "Internet Control Message Protocol." RFC 792, Settembre 1981, DOI 10.17487/RFC0792

[48] J. Turner, M. J. Schertler, M. S. Schneider, and D. Maughan, "Internet Security Association and Key Management Protocol (ISAKMP)." RFC 2408, Novembre 1998, DOI 10.17487/RFC2408

[49] N. Gupta, A. Saikia, and D. Sanghi, "Web application firewall", Indian Institute of Technology, Kanpur, vol. 61, Aprile 2007, p. 62. https://www.cse.iitk.ac.in/users/dheeraj/btech/namitg+abakashs.pdf

[50] M. Dermann, M. Dziadzka, B. Hemkemeier, A. Hoffmann, A. Meisel, M. Rohr, and T. Schreiber, "Best practices: use of Web Application Firewalls", The Open Web Security Application Project, OWASP Papers Program, Marzo 2008. https://owasp.org/www-pdf-archive/Best_Practices_WAF_v105.en.pdf

[51] J. Berner, "Building your own web application firewall as a service: And forgetting about", Giugno 2020. http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_069_Berner_WAF.pdf

[52] RedHat Inc, "Cosa significa open source?." https://www.redhat.com/it/topics/open-source/what-is-open-source, Ottobre 2019, Ultimo accesso il 10-09-2022

[53] NBS-System, "Repository pubblica di NAXSI WAF." https://github.com/nbs-system/naxsi, Ultimo accesso il 02-10-2022

[54] Zecure, "Repository pubblica di Shadow Daeamon WAF." https://github.com/zecure/shadowd, Ultimo accesso il 02-10-2022

[55] SpiderLabs, "Documentazione ufficiale ModSecurity WAF 2.x - 3.x." https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x)#Configuration_Directives, Ultimo accesso il 02-10-2022

[56] D. B. D. Aboba and J. Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile." RFC 3539, Giugno 2003, DOI 10.17487/RFC3539

[57] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Protocol Architecture." RFC 4251, Gennaio 2006, DOI 10.17487/RFC4251

[58] Google llc, "Documentazione minikube." https://minikube.sigs.k8s.io/docs/start/, Ultimo accesso il 15-10-2022

[59] Nginx Inc, "Documentazione nginx." https://www.nginx.com, Ultimo accesso il 15-10-2022