POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Integrazione e Implementazione di un Orchestratore di Sicurezza in Reti Virtualizzate

Relatori
Prof. Riccardo Sisto
Prof. Guido Marchetto
Dott. Fulvio Valenza
Dott. Jaloliddin Yusupov

Candidato
Cristian Gianetto
matricola: s265387

ANNO ACCADEMICO 2019-2020

Ringraziamenti

Quest'opera rappresenta la conclusione del mio lungo percorso di studi, che non avrei mai immaginato di essere in grado di sostenere, ma che è stato possibile solamente grazie al totale appoggio della mia famiglia, che mi ha sempre supportato e spronato nei momenti più difficili.

Credo che sia quindi doveroso ringraziare pubblicamente tutte le persone che mi hanno accompagnato in questo lungo viaggio, a partire dai miei affetti più vicini, quali mia mamma, mio papà e mia sorella, oltre ai miei cari nonni Alfonsina e Franco, che hanno sempre fermamente creduto in me, aiutandomi in ogni aspetto, sia morale che economico; sono davvero felice che finalmente possano festeggiare insieme a me, seppur in maniera molto limitata a causa di questa triste pandemia in corso, il risultato raggiunto.

Non posso poi non rivolgere un pensiero alla persona che per prima mi ha avvicinato al mondo dell'informatica, facendomelo conoscere ed amare fin da subito: oggi, purtroppo, non può più essere fisicamente qui con me, ma so' per certo che da lassù stai festeggiando, grazie Sergio!

Un ringraziamento particolare devo rivolgerlo al Prof.Ing. Vincenzo Borasi, ottima persona, grande conoscitore del mondo accademico, sempre disposta a dare il massimo per aiutare le persone a superare eventuali difficoltà fornendo ottimi e preziosi consigli.

Infine un grazie a tutti i parenti ed amici che mi hanno sempre sostenuto e che ci saranno sempre, ne sono certo, in futuro.

Indice

Capitolo 1

Introduzione

Fonti [1] [2] [3]

Con il passare degli anni l'importanza delle reti informatiche è divenuta sempre maggiore, facendo diventare la connettività un bisogno primario per chiunque. Mai questo concetto è stato più evidente che in questo triste anno (2020), segnato dalla pandemia da Covid-19 che ha costretto milioni di persone alla quarantena in casa propria, da cui comunque era necessario continuare la propria vita, nei limiti del possibile. Tutti gli studenti italiani avevano necessità di connettività per seguire le lezioni, i lavoratori necessitavano di connettività per continuare il proprio lavoro presso il loro domicilio, e tutti i cittadini, comunque, avevano necessità di accesso a internet per svolgere qualunque operazione, come ad esempio effettuare un pagamento o acquistare dei prodotti. Tutto questo si è tradotto in una notevole crescita di traffico sulle reti informatiche, portando alla luce anche i numerosi problemi legati a tale infrastruttura nella nostra penisola, ma ancor più grave ha causato una crescita preoccupante di cyberattacchi.

Un recente rapporto condotto da una società terza per conto di VMware[1] in Italia nel Giugno 2020, ha evidenziato proprio come gli attacchi informatici siano aumentati nel periodo pandemico, ma anche come questi attacchi siano sempre più evoluti.

L'indagine ha coinvolto circa 255 rappresentanti italiani di diverse aziende, operanti in svariati settori: servizi finanziari, sanità, pubblica amministrazione statale e locale, retail, progettazione industriale, alimentari e bevande, servizi di pubblica utilità, servizi professionali, media e intrattenimento. I primi dati significativi che rappresentano immediatamente l'urgenza dell'argomento (rappresentati nella Figura 1.1) sono i seguenti: il 98% dei professionisti della sicurezza ha affermato che c'è stato un aumento nel volume degli attacchi rispetto a Febbraio 2019 (aumento pari

[1] azienda nata nel 1998 con sede a Palo Alto, CA. E' specializzata nella produzione di software per ambienti virtuali.

Figura 1.1. Risultati analisi VMware Giugno 2020 - fonte: [1]

al 93%) e a Ottobre 2019 (aumento pari all'89%), e il 99% delle aziende intervistate ha dichiarato di aver subito una violazione a causa di un attacco informatico negli ultimi 12 mesi.

L'indagine si è poi focalizzata più precisamente sul periodo di chiusura forzata, cioè Marzo – Aprile 2020, per valutare come questo abbia inciso sulla sicurezza aziendale. I risultati evidenziano come, il fatto che i dipendenti abbiano forzatamente lavorato da casa, abbia causato un significativo aumento degli attacchi: ben il 90,5% delle aziende intervistate!

È quindi ormai evidente che è necessario pensare la sicurezza informatica come un qualcosa di indispensabile per qualsiasi realtà aziendale, non un qualcosa di facoltativo e rimandabile nel tempo, perché altrimenti questo errore potrebbe costare molto caro.

Le reti informatiche odierne si stanno trasformando, abbracciando sempre di più il mondo della virtualizzazione dei dispositivi e dei servizi di rete, aspetto che approfondiremo nel capitolo 2, dove si effettuerà anche un'analisi critica dei sistemi che sono oggi disponibili sul mercato, da cui possiamo già anticipare che non si è trovato un sistema che soddisfa i nostri standard di sicurezza, in quanto le informazioni fornite non garantiscono un intervento risolutivo al verificarsi di una violazione, ma in alcuni casi, si limitano a notificare tali eventi ad un componente esterno, senza fare proprio l'onore della sicurezza.

Il progetto ASTRID, di cui questa tesi fa' parte, ha come obbiettivo quello di cercare nuovi framework per la sicurezza informatica, al fine di migliorare la consapevolezza situazionale dei servizi virtualizzati, sia per ogni componente ma anche per il servizio nel suo insieme, e di facilitare (possibilmente automatizzare) il rilevamento e la reazione a sofisticati attacchi informatici, non solo provenienti dal mondo esterno, ma anche da attacchi interni al sistema stesso.

Gli obbiettivi specifici che si pone il progetto al fine di implementare i concetti espressi precedentemente sono i seguenti:

- **Separare la logica di business del servizio dalla gestione della sicurezza**: un generico servizio di rete, il quale svolge un determinato compito, può venire rappresentato come un grafo, in cui i nodi sono i singoli dispositivi che implementano il servizio e gli archi sono i collegamenti. Integrare i dispositivi di sicurezza nella definizione del grafo del servizio non è la soluzione migliore, in quanto solitamente richiede un intervento manuale. La sicurezza dovrebbe essere descritta definendo il "cosa" è necessario fare, e non il "come" farlo. Nel progetto ASTRID, quindi, le proprietà di sicurezza di ogni componente del grafo, nonché dell'intero servizio, sono definite da modelli e criteri che vengono utilizzati per configurare correttamente l'ambiente d'esecuzione. Lo sviluppatore specifica i requisiti di sicurezza e le politiche per la protezione del grafo;
- **Automatizzare la gestione della sicurezza e la risposta a minacce e ad attacchi**: questo è un punto fondamentale per gli obbiettivi di questa tesi. Oggi, molto spesso, le contromisure ad attacchi informatici arrivano in ritardo, quando ormai il danno è fatto, e questo avviene perché esse richiedono l'intervento umano. E' quindi necessario un nuovo framework, il quale distribuisca e configuri gli agenti di sicurezza secondo i modelli e le politiche definite dallo sviluppatore per intraprendere azioni di mitigazione automatiche (e quindi tempestive) in base alle strategie di sicurezza specificate. In ASTRID è quindi necessario implementare l'orchestrazione al fine di:
  - Interpretare automaticamente le politiche di sicurezza;
  - Configurare l'ambiente di esecuzione;
  - Raccogliere dati, misurazioni ed eventi sia da singole funzioni che da interi grafi di servizi;
  - Reagire a minacce e ad attacchi secondo strategie specifiche.
- **Ridurre l'overhead di runtime per l'elaborazione della sicurezza**:le applicazioni di sicurezza, richiedono operazioni di ispezione e monitoraggio che consumano risorse di calcolo e possono deteriorare le prestazioni generali del sistema, specialmente in ambienti virtuali dove l'accelerazione hardware è raramente disponibile, oltre ad essere anche loro dei possibili obbiettivi per gli attacchi. L'obbiettivo in ASTRID è quindi duplice:
  - Aumentare l'efficienza di elaborazione di (almeno) la parte più frequentemente eseguita delle attività di monitoraggio e rilevamento;
  - Proteggere le applicazioni che si occupano della sicurezza da attacchi.

  Per fare questo i dispositivi di sicurezza verranno suddivisi in due livelli: il piano dati e la logica di rilevamento, al fine di non sovraccaricare l'ambiente di esecuzione e per mantenere isolamento logico tra l'ambiente esterno e la logica di rilevamento, in modo da ridurre la superficie di attacco;

- **Supportare indagini legali e forensi in ambienti virtualizzati**: visto il crescente utilizzo degli ambienti virtuali, si prevede che esso sarà poi soggetto a utilizzo illegale. Sono quindi necessari dei meccanismi tecnici per facilitare le indagini forensi per l'identificazione di crimini e criminali. ASTRID supporterà indagini legali fornendo le seguenti caratteristiche tecniche:
  - Estrazione di flussi di traffico non crittografati dal grafico del servizio;
  - Accesso in tempo reale e offline a eventi, misurazioni e altre informazioni raccolte nel grafico del servizio.

Gli obbiettivi di questa tesi si focalizzano maggiormente sui primi 3 aspetti, e più specificatamente, sull'automatizzare la gestione della sicurezza e a fornire risposte in tempi rapidi ad attacchi. Ecco quindi che il nostro scopo sarà l'implementazione di un componente che farà parte dell'architettura del framework ASTRID: il Security Controller.

Capitolo 2

Software based network

In questo capitolo faremo dapprima un'analisi delle nuove tecnologie SDN ed NFV che vengono oggi molto utilizzate nel mondo delle reti, cercheremo di capire l'importanza di queste nuove tecniche e come possono essere sfruttate in questo ambito d'applicazione. Infine faremo una rapida carrellata di soluzioni MANO (MANagemente and Orchestrator) per architetture ETSI-NFV, prendendo in esame sia soluzioni open source (di cui abbiamo potuto conoscere con maggiore dettaglio la loro struttura di funzionamento), ma anche soluzioni proprietarie.

In tutti questi sistemi di orchestrazione che analizzeremo, si noterà che nessuno di essi si focalizza sull'aspetto della sicurezza, tranne in un caso (Open Source MANO), in cui però l'orchestratore non gestisce per conto proprio i segnali di allarme che riceve, ma si limita ad inviarli ad un componente esterno per la loro gestione.

2.1 Software Defined Network (SDN)

Nel mondo delle reti i vari dispositivi utilizzati(router, switch, NAT, ecc...), sono composti logicamente da 3 parti fondamentali:

- *Data plane o piano dati*: questa parte si occupa dell'inoltro/elaborazione dei vari pacchetti, la quale deve essere svolta nel modo più veloce possibile per non introdurre ritardi, ma anche perché ci sono molti pacchetti da gestire. Tipicamente le operazioni da svolgere sui pacchetti sono semplici (il look-up in una tabella, la variazione di un campo, il filtraggio, l'incapsulamento), ma tali operazioni vengono ripetute per ogni pacchetto che attraversa il dispositivo. Il flusso tipico del data-plane è il seguente: riceve › elabora › invia;

- *Control plane o piano di controllo*: questa è la parte atta a controllare il data plane, ovvero comprende tutte le funzioni che influenzano il modo di lavorare del data-plane stesso, come ad esempio i protocolli di routing;

- *Management plane o interfaccia di gestione*: questa parte è colei che si interfaccia con l'utente; comprende tutte le funzioni che vengono offerte all'amministratore della rete per controllare il dispositivo.

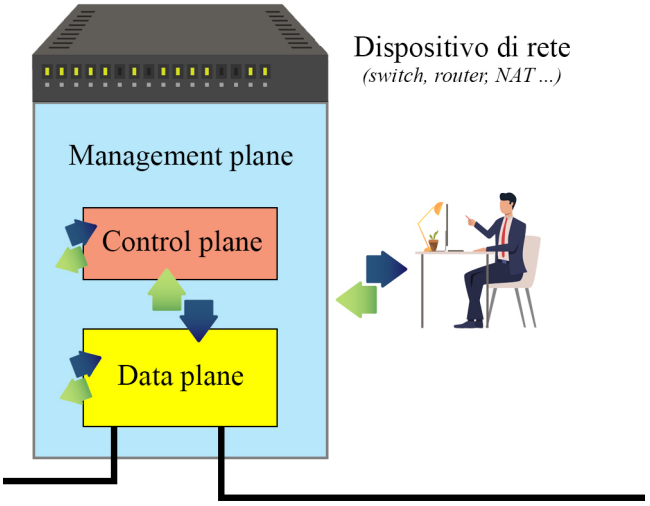


Figura 2.1. Schema di un generico dispositivo di rete

Queste parti logiche, però, formano un blocco unico in ogni dispositivo, e questo causa che ogni produttore di dispositivi di rete si deve implementare da sé tutti gli aspetti del dispositivo stesso (a partire dalla circuiteria, fino al software di gestione). Per capire meglio il problema che questo approccio causa, possiamo fare un confronto con quanto avviene nei PC: anche in un PC possiamo distinguere 3 parti logiche, l'hardware, il software di gestione e di interfacciamento, e il software applicativo. I produttori di personal computer, però, non implementano tutti e tre i livelli, tipicamente si limitano a fornire l'hardware, poi ci sono altri produttori che forniscono i software di gestione, e altri ancora che forniscono il software applicativo. Si voleva quindi arrivare a questa metodologia anche per i dispositivi di rete, in modo da favorire maggiormente lo sviluppo di nuove tecnologie in quanto, in questo modo, ogni produttore si può concentrare solamente sulla parte dei dispositivi di loro interesse: avremo quindi aziende che si specializzeranno in hardware dedicati per i vari dispositivi, altre aziende che si specializzeranno per fornire i migliori software di gestione, e così via.

Con l'acronimo SDN si indica un paradigma di rete, nato abbastanza recentemente, per cercare di adattare meglio il mondo del networking all'elevata dinamicità delle applicazioni odierne. L'innovazione fondamentale portata da questo paradigma è la separazione del piano di controllo (control plane) dal piano dati (data plane)

dei vari dispositivi di rete, permettendo quindi una centralizzazione delle attività di gestione e controllo dei vari dispositivi.

Già nel 2004 l'IETF propose un'interfaccia standard per il disaccoppiamento dei piani nei dispositivi, ma solo con la nascita del progetto "Ethane" dell'università di Stanford questa idea ha iniziato ad essere effettivamente applicata e studiata. Nella prima implementazione di Ethane, nel 2006, 19 switch erano gestiti da un unico controllore, il quale amministrava i flussi di più di 300 host connessi via cavo, oltre ad altri con connessioni wireless. Il progetto Ethane ha aperto la strada per la nascita di molti altri protocolli e tecnologie che sfruttano il paradigma SDN, uno fra tutti OpenFlow, celeberrimo protocollo molto utilizzato per accedere al piano dati di un dispositivo attraverso la rete.

Il disaccoppiamento e la centralizzazione delle attività di controllo dal piano dati, rende il lavoro di gestione dei vari dispositivi della reta molto più semplice ed agile, in quanto, se prima per ogni modifica (anche minima) era necessario agire fisicamente su ogni dispositivo con i notevoli costi che ciò comporta, ora grazie a questo paradigma, ciò non è più necessario, in quanto si ha un unico punto di controllo per tutti i vari dispositivi. La comunicazione fra l'SDN Controller e

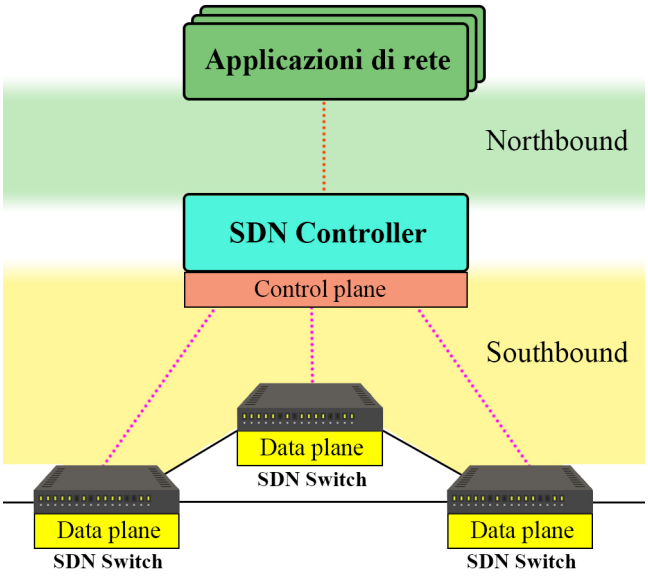


Figura 2.2. Architettura logica di una rete che utilizza il paradigma SDN

i vari dispositivi (che nella Figura 2.2 sono chiamati genericamente SDN Switch) avviene sull'interfaccia chiamata Southbound, e si sfruttano diversi protocolli, come NetConf o RestConf, ma il più utilizzato è OpenFlow. Tale interfaccia è comunque

di tipo eterogeneo, cioè il controller è in grado di comunicare con i vari dispositivi con più protocolli, in modo tale da poter supportare un gran numero di dispositivi diversi (anche di costruttori diversi).

L'SDN Controller comunica con i vari software che utilizzano la rete, attraverso l'interfaccia che prende il nome di Northbound. Le varie applicazioni di rete non conoscono i dettagli della rete sottostante, ma essa viene vista con l'astrazione "Big-switch", ovvero come se fosse un unico switch che permette lo scambio di pacchetti. È comunque possibile dividere la rete in "slices", ovvero fette, cioè partizioni di rete in modo che siano indipendenti l'una dalle altre. Ogni partizione avrà quindi il suo SDN Controller per la gestione di quella sezione.

Un aspetto fondamentale da considera è l'implementazione del controller, la quale può essere di 3 tipi:

- *Controller centralizzato*: abbiamo il controller che risiede in un unico server, il quale gestisce tutta la rete. È necessario fare in modo che questo server sia sempre operativo e performante, altrimenti l'intera rete manifesterà problemi;
- *Controllo logico centralizzato*: il controller è un sistema distribuito e ridondato su più macchine fisiche che collaborano fra loro. In questo modello, anche se un singolo server non è funzionante, la rete continuerà comunque a funzionare grazie alle altre macchine, che sopperiranno alla mancanza del server in down;
- *Controllo embedded*: in questa architettura ogni dispositivo integra al suo interno il controller, abbandonando quindi l'innovazione fondamentale portata dall'SDN. Il singolo controller, comunque, comunica con i suoi omologhi negli altri dispositivi. È possibile anche avere un controller centrale che influenza, dall'esterno, il controller interno al dispositivo.

Attualmente, la versione più utilizzata è il controllo logico centralizzato, in modo da garantire comunque l'operatività della rete anche nel caso in cui un singolo nodo di controller non sia disponibile.

Il protocollo principalmente utilizzato in questo ambito è OpenFlow, nato come prima tecnologia SDN, il quale può operare in 2 modi:

- Reattiva: la rete inizialmente non ha nessuna regola configurata, quindi tutti i pacchetti vengono inviati al controller, il quale prende le decisioni ed elabora le regole da far rispettare agli switch per i flussi successivi;
- Proattiva: Si configurano fin dall'inizio tutte le regole in ogni switch, di modo che la rete sappia già come comportarsi per ogni flusso che riceve. Quando un nuovo switch si connette, esso riceve già tutto il set di regole, pronto da essere applicato.

È possibile impostare delle policy molto fine sul controllore (ad esempio, è possibile distinguere anche le singole sessioni TCP, con i costi che comunque ne derivano), oppure più grossolane, a seconda delle necessità che si possono avere.

2.2 Network Function Virtualization (NFV)

Sempre più spesso, oggigiorno, si ha la necessità di compiere delle azioni sul traffico di rete, spesso azioni che siano trasparenti all'utilizzatore, come ad esempio, il filtraggio, il QoS, l'aggregazione o smistamento, ecc. Per ogni azione che è necessario compiere sul flusso di rete è possibile porre fisicamente un nuovo dispositivo all'interno della rete stessa che implementi la funzione desiderata, andando quindi a comporre la Service Function Chaining necessaria.

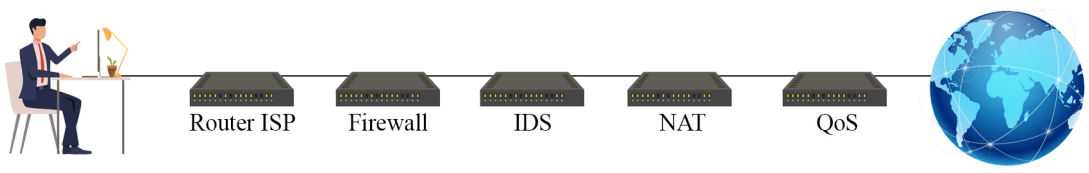


Figura 2.3. Schema di un generico servizio di rete

Tale approccio, che è quello classico, è però dispendioso sia in termini economici che di tempo, inoltre non permette di differenziare i flussi di traffico, ovvero, scegliere quali flussi devono attraversare specifici dispositivi (e quindi essere sottoposti a determinate azioni) e quali invece no, in quanto ogni pacchetto che attraversa quel tratto di rete subirà il medesimo trattamento di tutti gli altri.

Una prima risoluzione al problema della differenziazione del traffico, per fare in modo che solo determinati flussi subiscano determinate azioni, viene dallo sfruttamento del paradigma SDN: tutti i dispositivi vengono collegati ad un SDN switch centrale, il quale implementa solamente il data plane, mentre abbiamo un control plane centralizzato che imposta le regole in modo che solamente i flussi che necessitano di determinate azioni attraversano i dispositivi necessari, mentre per altri flussi su cui non è necessario eseguire quelle azioni, essi non attraverseranno il dispositivo in questione.

È possibile utilizzare anche un modo alternativo alle SDN, sfruttando le VLAN, ma questa soluzione non verrà approfondita ulteriormente in quanto non di nostro interesse. Ad ogni modo, le soluzioni viste fino ad ora, non risolvono tutti i problemi illustrati, in quanto è comunque necessario acquistare o comunque avere a disposizione il dispositivo fisico che implementi la funzionalità necessaria, con tutti i problemi che ne derivano: costi economici, affidabilità e mancata scalabilità.

Per questi scopi è nata la tecnologia NFV – Network Function Virtualization. La sua nascita è conseguente alle SDN, in quanto in NFV esse vengono sfruttate. Con questa tecnologia, anche nel mondo delle reti, si inizia a virare verso il virtuale, in quanto con l'acronimo NFV si intende lo sfruttamento di macchine generiche (ovvero server general-purpose), su cui si virtualizzano i vari dispositivi necessari al funzionamento della rete stessa. Questo porta notevoli vantaggi:

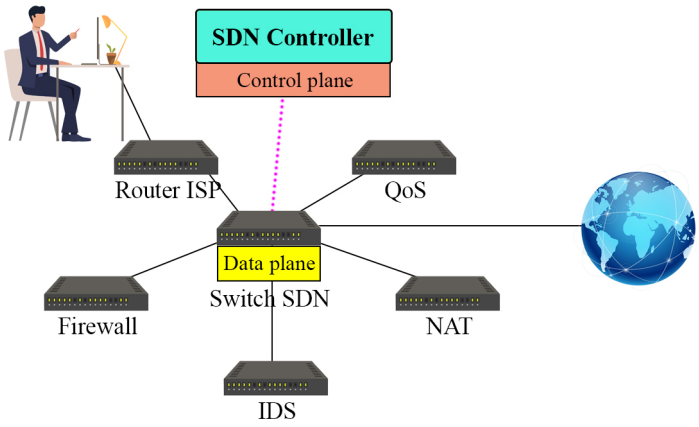


Figura 2.4. Service Function Chaining implementato con paradigma SDN

- Riduzione dei costi diretti e indiretti: non è più necessario acquistare apparecchiature dedicate per effettuare determinate operazioni, ma sono sufficienti server general-purpose, i quali generalmente sono meno costosi di apparecchiature dedicati; virtualizzando il dispositivo, non si ha più la necessità di raggiungere fisicamente il luogo dell'installazione e procedere con il setup, in quanto tutto è virtuale;
- Creazione dei servizi on-demand: se installiamo il dispositivo fisico nella rete, esso rimarrebbe sempre in funzione anche quando non si ha la necessità di utilizzarlo. Grazie alla virtualizzazione è possibile "creare" il dispositivo virtuale solamente quando si ha la necessità di utilizzarlo, e "distruggerlo" quando la necessità cessa; in questo modo si ottimizzano anche le risorse disponibili;
- Operazioni di aggiornamento apparecchiature effettuabili da remoto: se si ha la necessità di aggiornare un dispositivo, oppure sostituirlo, nel caso dei dispositivi fisici è necessario recarsi sul luogo in cui il dispositivo stesso è installato; in ambienti virtuali, invece, tutte queste operazioni possono essere compiute da remoto;
- Scalabilità: se le nostre esigenze di rete crescono, ad esempio abbiamo dei picchi di traffico in determinate ore della giornata, è sufficiente istanziare un numero maggiore di dispositivi virtuali in quelle ore, in modo che le richieste vengano gestite in un minor tempo. Trascorse le ore critiche, si torna ad avere il numero di apparecchiature tradizionale;
- Maggiore sfruttamento della potenza di calcolo: è possibile istanziare su un singolo server più dispositivi, ma anche istanziare i dispositivi a livello kernel, in modo da poter istanziare anche sul medesimo server delle VM per svolgere carichi di lavoro di tipo applicativo;

- Maggior affidabilità: Se abbiamo un caro di prestazioni su un server, oppure un guasto, possiamo spostare i dispositivi che venivano istanziati su di esso, su un altro server, in modo da garantire sempre le massime prestazioni possibili.

I vari dispositivi di rete, che fino ad ora erano apparecchiature fisiche dedicate, sono ora delle macchine virtuali (quindi del codice) che vengono eseguite su server generici.

Un aspetto da considerare è però l'efficienza, ovvero quanto la virtualizzazione sia efficiente rispetto allo sfruttamento del dispositivo fisico. È noto che per ottenere una prestazione identica ad un dispositivo fisico è necessario virtualizzare il dispositivo su più server, con tutte le conseguenze del caso, ovvero: un maggior consumo di elettricità, maggiori costi di climatizzazione, ecc.

Ci sono alcune tecniche che sono nate per cercare di aumentare le prestazioni dei dispositivi virtualizzati, come le DPDK (Data Plane Developed Kit) di Intel, o come Linux NAPI. Quest'ultimo è una modifica del framework di elaborazione dei pacchetti direttamente nei driver di dispositivi Linux, in modo da migliorarne le prestazioni e la velocità.

Un'alternativa possibile è implementare i vari servizi di rete a livello kernel, come già menzionato precedentemente, sfruttando la macchina virtuale eBPF. Tale VM è stata implementata nel 2013 come evoluzione della VM BPF, la quale aveva una serie di limitazioni che non permettevano di sfruttarla per questi scopi (come l'impossibilità di modifica il contenuto di pacchetti di rete). Questa VM permette l'esecuzione di funzioni (quindi, nel nostro caso, delle funzioni software che virtualizzano i dispositivi di rete) direttamente a livello kernel e non in user-space; in questo modo si massimizza l'efficienza.

Per gestire tutti i dispositivi necessari a fornire i servizi di rete, è necessario l'utilizzo di un orchestratore; i più comunemente utilizzati sono orchestratori che provengono anch'essi dal mondo virtualizzato: VMWare, OpenStack o Kubernetes.

Nel 2013, L'Istituto Europeo per gli Standard nelle Telecomunicazioni (ETSI) insieme alle principali industrie del settore, ha formato il gruppo ISG – Industry Specification Group, con lo scopo di creare un'architettura standard e linee guida generali valide in tutta Europa per la creazione di nodi di reti virtualizzate.

Nella Figura 2.5 è possibile osservare il framework proposto, composto dai seguenti blocchi:

- *Infrastruttura di virtualizzazione delle funzioni di rete (NFVI)*: esso rappresenta l'ambiente in cui vengono eseguite le funzioni virtualizzate. All'interno di questo blocco, troviamo le risorse fisiche (hardware):
  - Per eseguire l'elaborazione (CPU);
  - Il salvataggio di dati temporaneo (RAM);
  - Per la connettività (RETE);

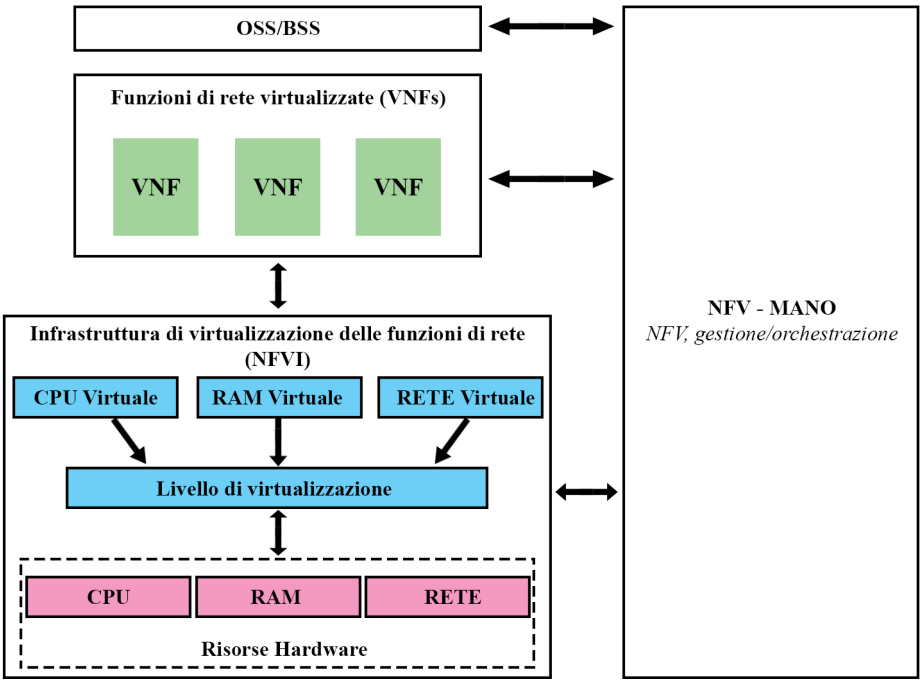


Figura 2.5. Schema NFV-ETSI

  - Al di sopra dei componenti fisici abbiamo un livello di astrazione, il quale esegue il disaccoppiamento fra le VNF e l'hardware d'esecuzione;
  - Al di sopra dello strato di virtualizzazione abbiamo le risorse hardware che vengono virtualizzate, pronte per essere utilizzate dalle VNF, quindi abbiamo la CPU virtuale, la RAM virtuale e la RETE virtuale.

- *Funzioni di rete virtualizzate (VNFs)*: In questo modulo abbiamo l'implementazione software dei vari dispositivi di rete, ovvero la virtualizzazione dei dispositivi che, fino ad ora, erano dispositivi fisici. Tali funzioni software sfruttano l'infrastruttura virtualizzata del modulo precedentemente descritto per essere eseguite, senza avere hardware dedicato;

- *NFV - MANO*: è il modulo che si occupa della gestione ed orchestrazione di tutte le risorse, interagendo con entrambe i moduli precedenti.

All'interno di quest'ultimo modulo abbiamo 3 componenti principali:

- *L'orchestratore NFV*: è il componente che ha il compito di coordinare i lavori di tutti gli altri componenti, interagendo anche con componenti esterni, come l'SDN controller;

- Il *VNF Manager*: è il modulo che si occupa del ciclo di vita delle varie VNF, ovvero della loro creazione (quando si ha la necessità di quel servizio all'interno della rete), del loro ridimensionamento e della loro terminazione;

- Il *gestore dell'infrastruttura virtualizzata*: componente che gestisce le risorse hardware fisiche e virtualizzate, ha una panoramica completa delle risorse disponibili, monitora le prestazioni e lo stato d'integrità di CPU e memoria. È quindi in grado di allocare, ridimensionare e rilasciare le risorse sfruttate dalle VNF, in modo da ottimizzarne l'utilizzo.

2.3 NFV - MANO: MANagement e Orchestration

Sempre di più, le reti odierne, integrano al loro interno funzioni di rete virtualizzate (NFV, illustrato nella sezione precedente), in quanto esse permettono di avere molti vantaggi sia sul piano della scalabilità e flessibilità, ma anche sul piano economico. Esistono, però, anche alcuni aspetti negativi che riguardano soprattutto la sicurezza, aspetto primario ai giorni nostro.

NFV permette di avere una velocità di adattamento al cambiamento dei vari scenari di rete, molto elevato rispetto ad una rete tradizionale, e questo si traduce nella necessità di avere un'orchestrazione altrettanto rapida. È quindi necessario scegliere un orchestratore adatto all'utilizzo in questi ambiti. Per questo motivo eseguiremo, in questa sezione, un'analisi sui possibili sistemi di orchestrazione oggi disponibili, ma anche in via di sviluppo, per l'ambito virtuale.

Come si è già visto nella figura 2.5, il modulo MANO (MANagement and Orchestration) comunica con tutti gli altri componenti dell'architettura NFV, diventandone quindi il cuore del sistema stesso, oltre che centro di controllo. La figura 2.6 è una variazione della 2.5 in cui viene rappresenta nel dettaglio l'architettura del modulo MANO, in cui possiamo distinguere 3 parti fondamentali:

- L'orchestratore NFV (NFVO – Network Function Virtualization Orchestrator);

- Il gestore delle VNF (VNF Manager);

- Il gestore dell'infrastruttura virtualizzata (Virtualized Infrastructure Manager – VIM).

Il componente NFVO svolge principalmente 2 compiti:

- Gestisce il ciclo di vita del Network Service: esso è il componente che si occupa di istanziare, modificare e terminare i servizi di rete, dialoga con il componente esterno OSS/BSS e può anche assolvere ad altri compiti personalizzati;

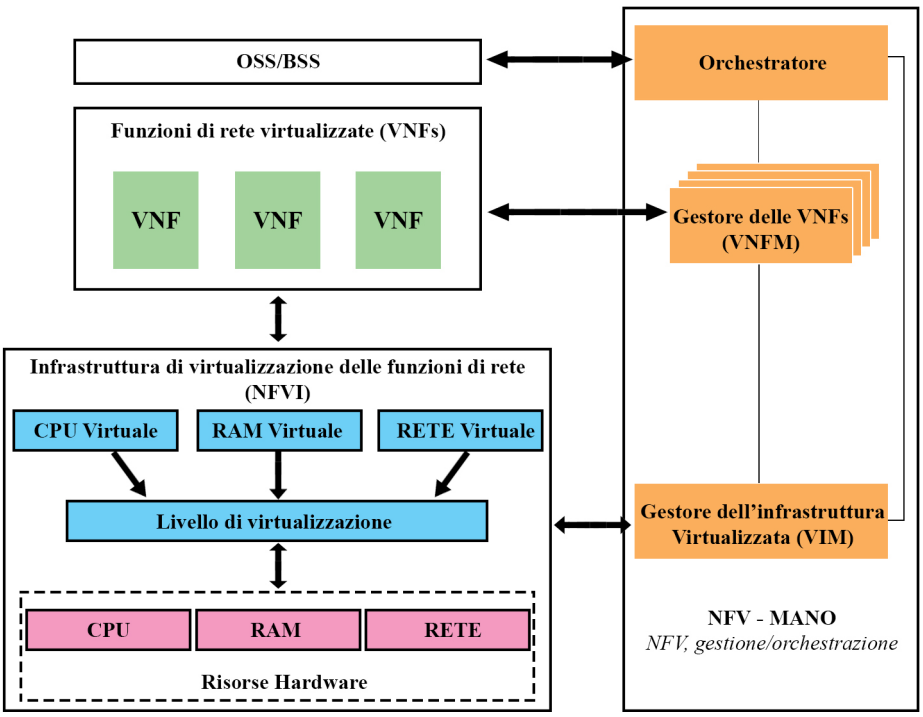


Figura 2.6. Schema NFV-ETSI con dettaglio del modulo MANO

- La gestione delle risorse disponibili nella NFVI (Network Function Virtualization Infrastructure): ogni richiesta di instanziazione di nuove funzioni deve essere autorizzata da questo componente, che successivamente si occupa anche della loro allocazione;

Il gestore delle VNF (VNFM – Virtualized Network Function Manager) ha il compito de gestire il ciclo di vita di una o più VNF; è quindi colui che si occupa di istanziare, modificare e terminare le funzioni di rete virtualizzate. È importante notare che spesso le operazioni che vengono svolte su una singola VNF vengono svolte anche su tutte le altre VNF che sono sotto il controllo del medesimo VNFM. Ci sono però alcune VNF che richiedono delle configurazioni particolari, diverse da tutte le altri ed è quindi necessario che il VNFM sia abbastanza flessibile per adattarsi alle varie VNF. Proprio per quest'ultimo motivo, è possibile che il gestore dell'infrastruttura decida di utilizzare più VNFM per gestire le varie VNF.

Infine, Il gestore dell'infrastruttura virtualizzata (VIM – Virtualised Infrastructure Manager) ha il compito di gestire le varie risorse (NFVI) necessarie poi alla successiva all'allocazione delle VNF svolta poi dal modulo NFVO descritto precedentemente. È possibile che il gestore dell'infrastruttura decida di utilizzare più

di una singola istanza del modulo VIM, creando quindi dei cosiddetti VIM specializzati, ovvero moduli dedicati alla gestione di uno specifico tipo di risorse (ad esempio, si potrebbe avere un VIM per la gestione delle risorse di rete, un altro per la gestione delle risorse di storage, ecc.). Il modulo VIM ha anche il compito di raccogliere e di notificare ai vari componenti dati sulle performance del sistema.

Possiamo ora vedere i principali moduli MANO ad oggi disponibili e che vengono comunque continuamente aggiornati. Molte soluzioni provengono dal mondo Open Source, ma ci sono anche molte soluzioni commerciali che evidenziano l'interesse del mercato per questo campo.

Per il mondo open source, analizzeremo i seguenti prodotti:

- Open Source MANO;
- OpenStack Tacker;
- Open Baton.

2.3.1 Open Source MANO

Fonte[4]

È un progetto coordinato dall'ETSI avviato nell'Aprile 2016. Questo progetto aggrega molti componenti già noti nel mondo open source, per cercare di ottenere un modulo di gestione ed orchestrazione compatibile con le specifiche ETSI. Il suo compito, come quello di qualunque altro orchestratore, è quello di occuparsi della gestione dei cicli di vita e delle configurazioni di tutte le VNF ospitate. Uno dei suoi obbiettivi è quindi quello di poter supportare il maggior numero di VNF, VIM e NFVI possibile.

Lo scopo principale di OSM è quello di creare un orchestratore di qualità, in grado di modellare e automatizzare i servizi utilizzate dalle aziende in ambito telco (Telefónica è una delle aziende partecipanti a questo progetto).

Lo sforzo d'integrazione di OSM MANO in un sistema NFV già esistente è ridotto al minimo grazie a 4 aspetti chiave:

- Un modello informativo conforme alle specifiche ETSI NFV, in grado di modellare e automatizzare l'intero ciclo di vita (dalla loro creazione, la loro gestione e monitoraggio, fino alla loro distruzione) delle funzioni di rete (sia virtuali, fisiche ma anche ibride), servizi di rete (NF – Network Functions) e segmenti di rete. Una delle capacità rilevanti di OSM MANO che realizza una delle promesse derivate da NFV è la capacità di istanziare reti su richiesta, per poter essere sfruttate immediatamente, oppure essere rivendute a terzi;
- La presenza di un'interfaccia di comunicazione unificata (NBI – North Bound Interface) sulla quale sono esposte delle API che forniscono le astrazioni necessarie per il pieno funzionamento dei servizi e delle funzioni di rete, oltre

che dei segmenti di rete, sotto il suo controllo, evitando quindi l'esposizione di informazioni non necessarie;

- Il concetto esteso di servizio di rete (NS – Network Service), che in OSM rappresenta l'elemento costitutivo minimo per la gestione delle reti come servizio, all'interno del quale si raggruppa in un unico oggetto un insieme di funzioni di rete interconnesse, che possono essere sia virtuali, che reali ma anche ibride, e che possono avere diverse ubicazioni (centralizzate o meno) e anche diverse aree geografiche. Il tutto viene controllato da OSM che gestisce quindi l'intero ciclo di vita del NS;

- OSM è in grado anche di assumere il ruolo di Slice Manager nel caso di reti stratificate, gestendo il ciclo di vita delle varie Slice di rete (può essere sfruttato su casi d'uso legati alla tecnologia 5G).

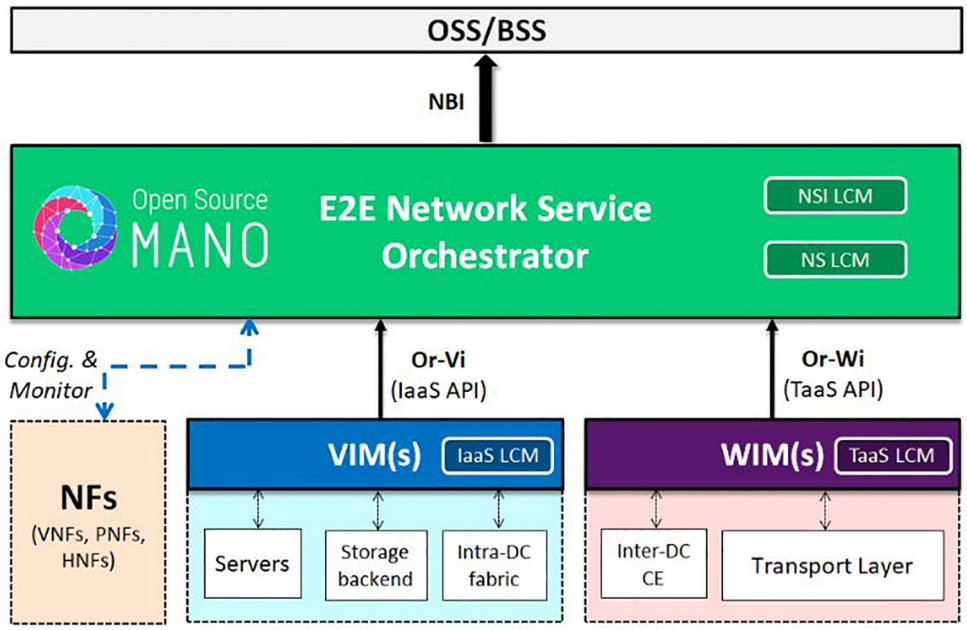


Figura 2.7. Schema d'integrazione di Open Source MANO in un'architettura NFV esistente - fonte: [4]

Nella figura 2.7 possiamo vedere uno schema di come OSM può essere integrato in un'architettura NFV già presente.

Dal punto di vista della sicurezza, MANO offre un livello di garanzia pari ad altre applicazioni oggi sul mercato, in particolare sull'interfaccia NBI fornisce autenticazione e accesso autorizzato ai client basato su TLS, inoltre segue le best practices comuni, quali: nessuna password viene salvata, le credenziali di accesso ai

database interni non sono quelle di default, criptatura delle informazioni sensibili fornite, nessun processo in esecuzione come utente root nell'host di sistema ed infine, la comunicazione fra moduli avviene in maniera protetta e autenticata se sono coinvolti server diversi.

MANO invia a sistemi esterni, i LOG e i segnali di allarme, i quali devono implementare la logica di gestione degli stessi.

2.3.2 OpenStack Tacker

Fonte[5]

Tacker è un progetto ufficiale di OpenStack, nato da una diramazione del progetto Neutron, ha assunto il nome di Tacker nel 2015. Tale progetto ha come obbiettivo lo sviluppo di un VNF Manager (VNFM) generico e un orchestratore NFV (NFVO) per la gestione delle VNF su un'architettura NVF come OpenStack[1].

Anche Tacker è conforme allo standard ETSI MANO ed è in grado di gestire i servizi di rete in maniera end-to-end.

OpenStack Tacker fornisce un blocco integrato formato da NFVO ed VNFM, ma supporta anche VNFM esterni. Nell'architettura di OpenStack Tacker mo-

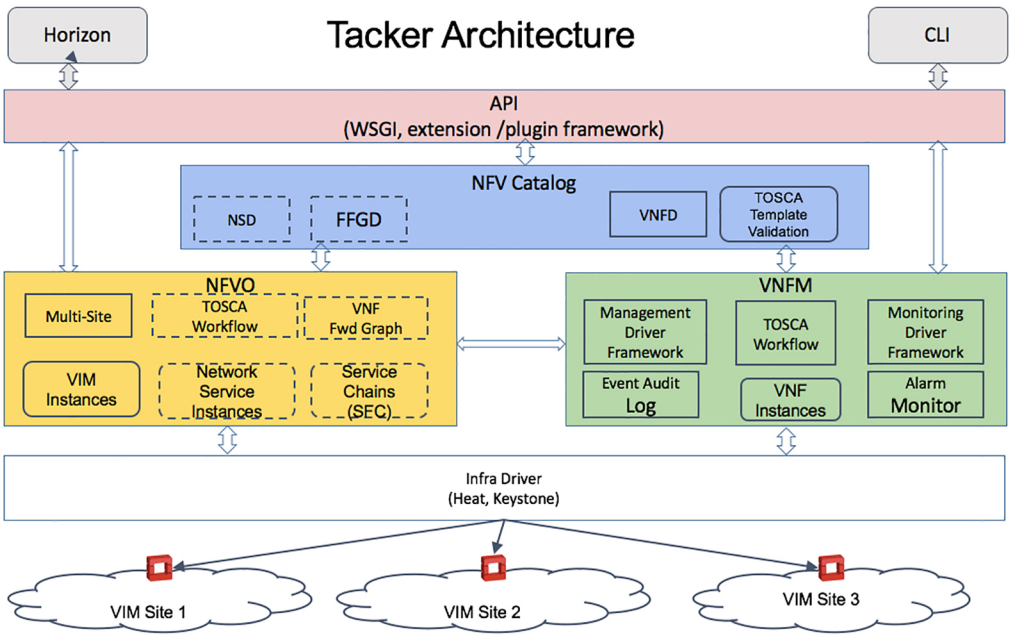


Figura 2.8. Architettura di OpenStack Tacker - fonte: [5]

[1]OpenStack è un controller opensource con licenza Apache per la creazione e la gestione di sistemi cloud virtuali.

strata nella figura 2.8 vediamo che è possibile gestire OpenStack Tacker tramite l'interfaccia standard di OpenStack (Horizon) oppure tramite l'interfaccia a riga di comando (CLI – Comman Line Interface). L'interfacciamento con le VIM avviene tramite l'interfaccia Infra Driver, sfruttando Heat[2]

Tacker, di per sé, è composto da un catalogo di risorse (NFV Catalog), un orchestratore di risorse (NFVO) e da un VNF Manager (VNFM). Infine, c'è da notare che ogni operazione che viene svolta è autorizzata (o negata) da Keystone, noto come Identity Service, il quale si occupa di gestire gli utenti ed i gruppi e le loro autorizzazioni all'interno dell'intero sistema, creando i token necessarie alle API per essere eseguite.

2.3.3 Open Baton

Fonte [7]

Open Baton è un progetto tedesco, disponibile con licenza apache su GitHub. Si basa anch'esso sullo schema architetturale ETSI MANO, e fornisce un blocco NFVO, un generico VNFM (ma supporta anche VNFM specifici), supporta più VIM OpenStack oltre a un meccanismo per altre VIM.

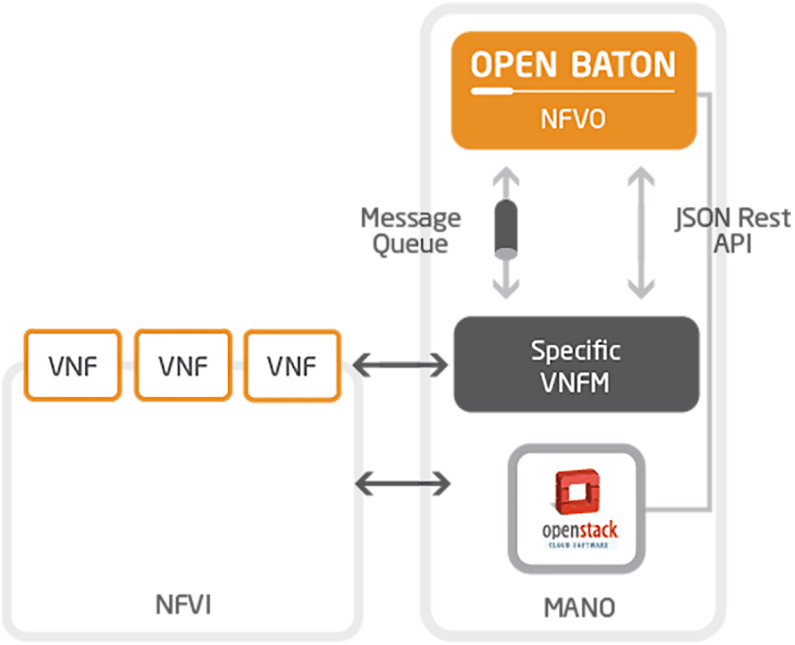


Figura 2.9. Architettura di Open Baton - fonte: [7]

[2]è un servizio per orchestrare applicazioni cloud utilizzando un modello dichiarativo tramite API REST nativa di OpenStack. Fonte [6]

Nella figura 2.9 possiamo vedere l'architettura di Open Baton.

Il progetto Open Baton è sviluppato in Java con il framework Spring.io (il medesimo framework che abbiamo utilizzato anche nel nostro progetto dell'Astrid Controller, e di cui faremo un approfondimento nel capitolo3.2.

Open Baton è composto da diversi moduli software e plugin, i quali offrono ognuno le loro caratteristiche per implementare una specifica funzione: abbiamo Zabbix per il monitoraggio, RabbitMQ come message brocker per la comunicazione fra i vari moduli e altri. Tali moduli, comunque, possono venire omessi, sia in tutto che in parte a seconda delle esigenze.

Moduli MANO proprietari

Passiamo ora ad analizzare alcuni moduli MANO proprietari, di cui andremo poi ad illustrarne le caratteristiche, anche se con meno dettagli rispetto alle versioni open source in quanto sono, appunto, soluzioni proprietarie. I moduli che andremo ad analizzare sono rappresentati nella Tabella 2.1.

Tabella 2.1. Moduli MANO proprietari analizzati

Cisco	Network Services Orchestrator (NSO) e Elastic Services Controller (ESC)
Ericsson	Ericsson Orchestrator
ZTE	CloudStudio

2.3.4 CISCO - Network Services Orchestrator (NSO) e Elastic Services Controller (ESC)

Fonte [8][9]

Cisco Elastic Services Controller (ESC) e Network Services Orchestrator (NSO) sono due moduli proprietari Cisco che implementano, rispettivamente, il Virtual Network Functions Manager (VNFM) e l'NFVO insieme all'OSS. L'ESC gestisce le VNF fornendo i servizi virtuali e monitorandone lo stato e il carico. È in grado, in base a regole definite, di eseguire la scalabilità delle VNF stesse sia in entrata che in uscita, oltre a supportare il ripristino automatico delle VM, quando esse si arrestano in modo anomalo.

L'ESC funziona come una macchina virtuale installabile su tutti i servizi maggiormente diffusi al giorno d'oggi, e fornisce un framework integrato di strumenti che seguono l'intero ciclo di vita di una VNF, il quale si compone delle fasi seguenti(raffigurate in Figura 2.10):

Figura 2.10. Schema logico del componente ESC - fonte: [8]

- *Onboarding*: questa è la fase della preparazione dell'immagine VNF, la definizione dei requisiti, delle metriche e delle soglie di monitoraggio, criteri di posizionamento delle VM e regole di scalabilità;

- *Deploy*: fase di caricamento dell'immagine VNF, in cui avviene anche l'applicazione delle regole di affinità, oltre alle soglie di monitoraggio e di qualsiasi altro parametro;

- *Monitor*: fase di monitoraggio in cui si controlla, per ogni VNF una serie di parametri, come il ping, l'utilizzo della memoria, l'utilizzo delle risorse di calcolo, ed il bit rate sia in input che in output. Viene anche permesso all'utente di definire delle metriche personalizzate che quindi anch'esso verranno monitorate in questa fase;

- *Scale*: fase di ridimensionamento delle VNF in base al crescere o al diminuire della domanda;

- *Heal*: monitoraggio dell'integrità delle VM con la funzionalità del ripristino in caso di anomalie;

- *Update*: fase di aggiornamento della distribuzione, che può includere VM, risorse, configurazioni, indirizzamento IP e immagine della VNF;

- *Undeploy*: fase di cessazione del sevizio offerta dalla VNF e smontaggio dell'immagine.

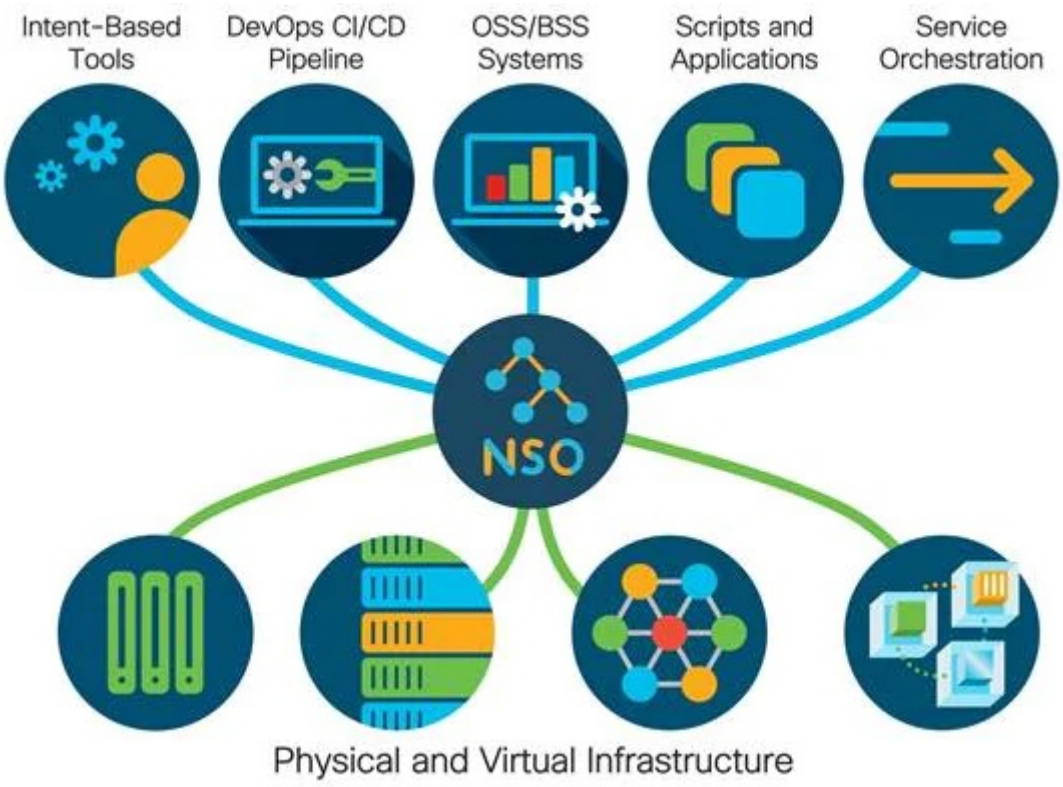


Figura 2.11. Schema logico del componente NSO - fonte: [9]

Come si può vedere dalla figura 2.11, l'NSO fornisce un ponte di collegamento fra i framework di automazione ed orchestrazione e l'infrastruttura (sia fisica che virtuale che ibrida) sottostante.

L'NSO ha 3 componenti architetturali:

- Interfaccia programmatica basata su un modello che permette il controllo di qualsiasi cosa, dalla sua istanziazione fino alla sua terminazione (ovvero per l'intero ciclo di vita);
- Un archivio dati scalabile che ospita le varie configurazioni;
- Un livello di astrazione del dispositivo che viene utilizzato per mediare l'accesso sia al dispositivo fisico che virtuale, il quale può essere sia un dispositivo Cisco, ma anche di altri produttori;

2.3.5 ERICSSON - Ericcson Orchestrator

Fonte [10]

Ericsson Orchestrator, nato con il nome di Cloud Manager e successivamente ribattezzato, è uno dei principali componenti del prodotto di Ericsson Dynamic Orchestration. Questa soluzione interpreta il ruolo di un orchestratore NFV (NFVO) e di un generico VNF manager (Generic VNFM) occupandosi dell'intera gestione del ciclo di vita VNF, oltre ad essere anche un Service Orchestration (SO) e anche un Transport Orchestration (WANO).

Si basa anch'esso sull'architettura ETSI NFV.

2.3.6 ZTE - CloudStudio

Fonte [11]

CloudStudio di ZTE è un progetto che coinvolge 3 prodotti differenti:

- CloudStudio GSO;
- CloudStudio NFVO;
- CloudStudio VNFM;

CloudStudio GSO è un sistema OSS per le reti virtualizzate, che fornisce una soluzione end-to-end per la progettazione, l'orchestrazione, la configurazione, l'attivazione, l'inventario e la manutenzione delle sezioni di rete.

CloudStudio NFVO rappresenta l'orchestratore dell'architettura NFV, in grado anche di cooperare con software di terze parti.

CloudStudio VNFM realizza la gestione automatica dell'intero ciclo di vita delle VNF evitando l'intervento manuale il quale potrebbe introdurre errori. Più precisamente, si occupa della creazione delle istanze, il monitoraggio, l'espansione, la riparazione, l'aggiornamento e il backup. Integra al proprio interno la gestione unificata di VNF di terze parti, in modo da fornire più servizi di rete.

Capitolo 3

Strumenti utilizzati

In questo capitolo faremo un'analisi dei principali strumenti che vengono utilizzati per lo sviluppo dell'Astrid Security Controller

3.1 Progetto Apache Maven

Apache Maven è uno dei tool più famosi per la gestione di progetti Java, il quale si occupa di semplificare diversi aspetti relativi alla scrittura del codice, come:

- Gestione delle librerie utilizzate;
- Documentazione;
- Utilizzo di plugin per l'ampliamento delle funzioni offerte;
- Esportazione automatizzata del progetto;
- Fase di testing;

La parte centrale di un progetto Maven è il file pom.xml. Tale acronimo indica Project Object Model, ovvero un file xml il quale ha lo scopo di accogliere tutte le informazioni di configurazione sul progetto Java che si sta sviluppando, come le dipendenze dalle librerie, i plugin sfruttati e molto altro.

Una delle forze di Maven è la possibilità di lanciare la compilazione e l'esecuzione del proprio progetto Java utilizzando un semplice comando da riga di comando; sarà Maven che si occuperà di ottenere le dipendenze necessarie all'esecuzione del programma

3.2 Spring Boot Framework

Fonti [12] [13] [14]

Sempre più spesso, la scrittura di nuovi software non avviene mai da zero, ma ci si appoggia su strutture già predefinite, in modo da semplificare il lavoro del programmatore e di ridurre i tempi di sviluppo, evitando di scrivere più volte il medesimo codice in più progetti e cercando di ottimizzarlo.

L'utilizzo di framework, che sono disponibili per quasi tutti i linguaggi utilizzati oggigiorno, permette di avere una base di partenza già scritta, su cui costruire le nuove funzioni che implementano il comportamento specifico del software. Tali framework, mettono a disposizione delle classi astratte o API le quali implementano i comportamenti standard di tutte le applicazioni (ad esempio, la Standard Template Library – STL in C è un framework fondamentale per lo sviluppo di applicazioni in C, la quale permette di avere a disposizione già tutta una serie di strutture dati complesse, algoritmi e iteratori), in questo modo il programmatore può concentrarsi sulle funzioni peculiari del software in questione.

Sono nati moltissimi framework, alcuni sono generali (come la STL in C, precedentemente nominata), altri invece sono più specifici per determinati scopi, ovvero offrono una serie di funzioni al programmatore, che sono tipiche per raggiungere determinati obbiettivi.

In quest'ottica, per lo sviluppo del Security Controller di ASTRID si è scelto di utilizzare il framework Spring Boots, il quale offre una serie di vantaggi e semplificazioni:

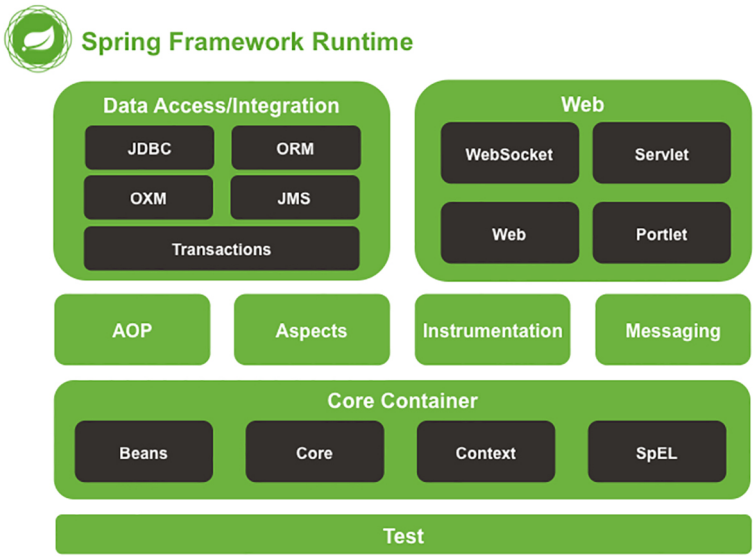


Figura 3.1. Moduli che compongono il framework Spring Boot - fonte: [14]

- *Modularità*: il framework Spring Boots è composto da circa 20 moduli (rappresentati nella Figura 3.1). Ogni modulo fornisce delle semplificazioni per determinati ambiti, ad esempio: i moduli che sono raggruppati nel Data Access/Integration forniscono delle API che permettono di accedere ai dati e di manipolarli in un database (JDBC), in un XML (OXM), in messaggi java (JMS) e anche per la gestione delle transazioni (Transactions), in modo molto più semplice, evitando al programmatore l'onere di gestire ogni singolo aspetto dell'operazione. Il vantaggio del framework Spring è quindi che è possibile sfruttare solamente i moduli che sono necessari al programma in oggetto, senza caricare inutilmente tutti i moduli del framework per poi sfruttarne solo una parte. È quindi possibile anche adottare un approccio incrementale.

- *L'inversion of control (IoC)*: il framework Spring implementa questo principio architetturale. Nella programmazione procedurale tradizionale, tipicamente, il codice applicativo per svolgere le sue azioni richiama procedure di libreria (passandole quindi il controllo; ad esempio, nel linguaggio C, per acquisire un carattere digitato da tastiera si può usare il comando scanf, il quale appartiene alla libreria standard, e che non fa altro che richiamare la funzione di sistema che acquisisce il carattere). Nei software che implementano il pattern IoC, invece, il codice applicativo riceve il controllo da un componente appartenente ad una libreria. Grazie a questo pattern, quindi, non è più il programmatore che deve definire la logica del flusso di controllo, ovvero le operazioni di creazione ed inizializzazione degli oggetti e l'invocazione dei metodi, ma sarà il framework stesso che, reagendo a "stimoli" se ne occuperà;

- *Dependency Injection (DI)*: questo pattern di programmazione consiste nel fatto che oggetti complessi non istanzieranno mai altri oggetti all'interno dei loro metodi o nel proprio costruttore, ma le dipendenze con altre classi saranno fornite dall'esterno, utilizzando i parametri del costruttore, oppure iniettate tramite metodi setter. Il framework Spring facilità il compito del programmatore: l'applicazione viene eseguita all'interno di un ambiente che prende il nome di IoC Container, il quale si occupa di creare e iniettare le varie dipendenze necessarie alle classi dell'applicazione per il suo funzionamento, basandosi sulle configurazioni definite dall'utente, generalmente utilizzando file XML;

- *Embedded Server nativo*: il framework Spring Boots crea un file JAR, all'interno del quale, oltre alla nostra applicazione, avremo già anche l'ambiente operativo adatto: l'embedded server. Questa caratteristica facilità di molto la creazione di applicazioni web (com'è la nostra) e il testing, in quanto sarà risparmiato tutto il tempo necessario a istanziare Tomcat o JBOSS.

3.3 Drools Rules

Fonti [15] [16] [17]

In molte applicazioni che vengono utilizzate in ambito aziendale è necessario definire delle regole a seguito delle quali si devono verificare determinati eventi. Un semplice esempio per chiarire il concetto può essere un software responsabile del calcolo dello sconto su un determinato articolo di vendita: ipotizziamo che un negozio di elettrodomestici decida di effettuare una campagna pubblicitaria che prevede di applicare le seguenti scontistiche in base alla categoria merceologica dell'articolo (le scontistiche sono rappresentate nella Tabella 3.1).

Tabella 3.1. Percentuali di sconto applicate alle categorie merceologiche

Categoria	**% di sconto**
Frigoriferi	10
Congelatori	5
Lavatrici	20
TV	15
Lavastoviglie	20
Piani Cottura	10
Cellulari	15

È ipotizzabile modificare le funzioni che partecipano al calcolo dello scontrino, inserendo dei costrutti if, oppure un costrutto switch, come rappresentato nel Listato 3.1.

Tale modifica, pur essendo corretta e lecita per gli scopi che vogliamo raggiungere, non è però ottimale, in quanto se il negoziante decide di modificare le percentuali di sconto per una o più categorie merceologiche, oppure vuole eliminare uno sconto su una categoria piuttosto che inserirne di nuovi, ha necessità che lo sviluppatore del software agisca sul codice per effettuare le modifiche necessarie.

Proprio per cercare di separare la scrittura delle regole di funzionamento dei programmi, dalle funzioni logiche, sono nati i Business Rule Management System – BRMS, di cui Drools fa' parte. Drools è un progetto open-source, scritto in Java, che si occupa di definire, eseguire, monitorare e mantenere la logica decisionale all'interno di un'organizzazione.

Per il suo funzionamento, Drools, si basa sui dati disponibili (che prendono il nome di facts) e sulle regole che sono state definite; è presente un "motore di regole" il quale, al verificarsi di determinate condizioni che corrispondono a regole imposte, lancia l'azione corrispondente. Lo schema di funzionamento è rappresentato nella Figura 3.2.

Le regole vengono scritte in appositi file *.drl*, e la sintassi è rappresentata nel Listato 3.2.

```
switch(Categoria){
    case "Frigoriferi":
        sconto_percentuale = 10;
        break;
    case "Congelatori":
        sconto_percentuale = 5;
        break;
    case "Lavatrici":
        sconto_percentuale = 20;
        break;
    case "TV":
        sconto_percentuale = 15;
        break;
    case "Lavastoviglie":
        sconto_percentuale = 20;
        break;
    case "Piani cottura":
        sconto_percentuale = 10;
        break;
    case "Cellulari":
        sconto_percentuale = 15;
        break;
    default:
        sconto_percentuale = 0;
}
```

Listato 3.1. Costrutto switch per il calcolo dello sconto percentuale in base alla categoria merceologica

Listato 3.2. Scheletro di una regola Drool

```
rule [nome_regola]
    when
    [condizioni]
    then
    [azioni]
end
```

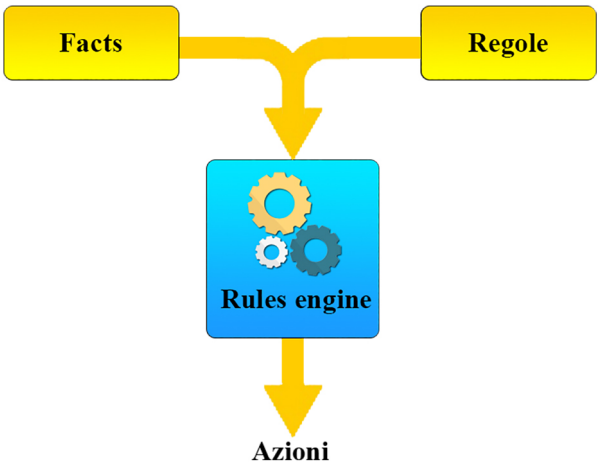


Figura 3.2. Schema di funzionamento di Drools Engine

Riprendendo il nostro esempio iniziale, quindi, ipotizzando che abbiamo una classe "articolo", la quale rappresenta tutti gli articoli che abbiamo in vendita e che contiene un campo di tipo String "Categoria", la quale contiene il nome della categoria merceologica a cui l'articolo si riferisce, oltre ad un campo di tipo int "ScontoPercentuale" che contiene la percentuale di sconto associata all'articolo, le Drools Rules sono quelle rappresentate nel Listato 3.3.

Se in futuro sarà necessario eseguire qualche modifica sulle regole imposte, il negoziante non avrà quindi più necessità di interpellare lo sviluppatore del software per modifica il codice, ma è sufficiente modificare la regola corrispondente all'interno del file *.drl*.

	A	B	C	D
1	**RuleSet**	**com.gestshop.drools**		
2	**Import**	com.gestshop.drools.model.Article, com.gestshop.drools.model.Article.CategoryList		
3				
4	**RuleTable** Discount rates			
5	**NAME**	**CONDITION**	**ACTION**	**ACTION**
6		$oggetto:articolo		
7		$oggetto.getCategory() in ($param)	$oggetto.setScontoPercentuale($param);	System.out.println("L'articolo è scontato del " + $param + "%");
8	NAME	Categoria Merceologica	Sconto %	Sconto %
9	offerta su frigoriferi	Frigoriferi	10	10
10	offerta su congelatori	Congelatori	5	5
11	offerta su lavatrici	Lavatrici	20	20
12	offerta su TV	TV	15	15
13	offerta su lavastoviglie	Lavastoviglie	20	20
14	offerta su piani cottura	Piani cottura	10	10
15	offerta su cellulari	Cellulari	15	15

Figura 3.3. Definizione delle regole Drools Rules mediante Microsoft Excel

È anche possibile generare automaticamente il file *.drl* a partire da tavole di decisione che è possibile scrivere, ad esempio, in Microsoft Excel, in questo modo

Listato 3.3. Drools Rules d'esempio

```
rule "offerta su frigoriferi"
    when
    $oggetto: articolo( categoria == "Frigoriferi")
    then
    $oggetto.setScontoPercentuale(10);
    System.out.println( "L'articolo risulta scontato del 10%" );
end

rule "offerta su congelatori"
    when
    $oggetto: articolo( categoria == "Congelatori" )
    then
    $oggetto.setScontoPercentuale(5);
    System.out.println( "L'articolo risulta scontato del 5%");
end

rule "offerta su lavatrici"
    when
    $oggetto: articolo( categoria == "Lavatrici" )
    then
    $oggetto.setScontoPercentuale(20);
    System.out.println( "L'articolo risulta scontato del 20%");
end

rule "offerta su TV"
    when
    $oggetto: articolo( categoria == "TV" )
    then
    $oggetto.setScontoPercentuale(15);
    System.out.println( "L'articolo risulta scontato del 15%" );
end

rule "offerta su lavastoviglie"
    when
    $oggetto: articolo( categoria == "Lavastoviglie")
    then
    $oggetto.setScontoPercentuale(20);
    System.out.println( "L'articolo risulta scontato del 20%");
end

rule "offerta su piani cottura"
    when
    $oggetto: articolo( categoria == "Piani cottura")
    then
    $oggetto.setScontoPercentuale(10);
    System.out.println( "L'articolo risulta scontato del 10%");
end

rule "offerta su cellulari"
    when
    $oggetto: articolo( categoria == "Cellulari" )
    then
    $oggetto.setScontoPercentuale(15);
    System.out.println( "L'articolo risulta scontato del 15%" );
end
```

si facilita ulteriormente il lavoro a personale non tecnico (Figura 3.3). Tali tavole di decisione è possibile salvarle in file XLS ma anche in file CSV.

Esistono, infine, anche dei meccanismi che permettono la scrittura automatica delle regole a partire da un semplice testo scritto in linguaggio comune.

Le azioni che possono essere intraprese da una regola Drools vengono scritte in linguaggio Java, ma ci sono anche alcuni comandi specifici da poter utilizzare, quali:

- *insert()*: comando che serve per inserire all'interno del motore di regole degli oggetti;
- *update()*: comando che aggiorna lo stato di un oggetto all'interno del motore delle regole;
- *modify()*: comando che modifica lo stato di un oggetto che si trova all'interno del motore delle regole;
- *retract()*: comando utilizzato per eliminare un oggetto che si trova all'interno del motore delle regole.

È possibile utilizzare questi comandi per mantenere uno stato degli oggetti, in modo da utilizzarlo per confronti futuri.

Un ultimo aspetto importante da notare è la possibilità di legare il motore delle regole a delle sessioni (KieSession), le quali rappresentano un'istanza di esecuzione del motore delle regole, il quale ha una determinata configurazione e un preciso insieme di regole. In un progetto Java, è quindi possibile avere più sessioni che istanziano diversi motori di regole.

Riassumendo, quindi, l'introduzione di Drools nel progetto permette di apportare le seguenti migliorie:

- Separazione fra la logica dell'applicazione e le regole aziendali: le regole su cui il programma lavora sono completamente separate dal codice del programma stesso, in questo modo la manutenibilità del software è maggiore e senza il rischio di manipolare regole aziendali;
- Velocità e scalabilità: si eliminano dalla logica del software tutti i blocchi if o costrutti switch necessari per il controllo delle azioni di intraprendere, in quanto vengono sostituiti dalle regole di Drools;
- Centralizzazione delle regole aziendali: tutte le regole aziendali sono concentrate in un unico repository, in questo modo si sa sempre dove agire quando necessario;
- Regole comprensibili a tutti: le regole sono scritte in linguaggi più vicini a quello colloquiale, quindi facilmente comprensibili anche a personale non tecnico. Sono inoltre disponibili alcune alternative che facilitano la scrittura di

tali regole (come le tavole di decisione oppure l'utilizzo di tool che estrapolano le regole da testo scritto in linguaggio comune);

3.4 Apache Kafka

Fonte [22]

Nel mondo odierno siamo ormai circondati di applicativi software che svolgono svariate funzioni, dai pagamenti digitali alla prenotazioni di visite, vacanze e molto altro. Tutti questi servizi, i quali sono servizi distribuiti e che devono essere in grado di processare dati che giungono simultaneamente da più postazioni, richiedono una comunicazione con svariati altri applicativi software, pensiamo ad esempio alla prenotazione di una visita ospedaliera, nel momento in cui si fissa il giorno e l'ora è necessario accertarci che nessun altro abbia prenotato quel medesimo slot, e di inserire la prenotazione all'istante per evitare che altri possano confermare il medesimo appuntamento.

Ci sono molti modi possibili per permettere la comunicazione fra moduli indipendendenti, uno fra tutti è proprio il sistema Apache Kafka, che è ciò che abbiamo scelto di utilizzare per l'elaborazione del Security Controller.

Apache Kafka è una piattaforma di streaming distribuito nonchè di messaggistica in real-time; nata nel 2011 per rispondere alle esigenze di LinkedIn, essa offre una soluzione con un throughput e una scalabilità elevati, oltre ad un alto livello di affidabilità grazie alla replica dei dati. In un architettura che utilizza Apache Kafka, i vari moduli possono avere 2 ruoli:

- Produttore: sono i moduli che elaborano e producono dati e che devono essere resi disponibili ad altri moduli;
- Consumatori: essi sono i moduli che sono in ascolto e che ricevono i messaggi.

Apache Kafka è eseguito come cluster su server (che possono essere singoli o multipli) e ogni flusso di dati in entrata viene salvato in modo durevole all'interno di *topic*, i quali possono ancora essere suddivisi in partizioni, come viene mostrato dalla Figura 3.4.

Affinchè un consumatore possa leggere i messaggi di un determinato *topic* è necessario che si registri. In Apache Kafka i produttori e i consumatori non hanno alcun legame diretto fra loro, quindi i produttori, ad esempio, non è necessario che attendano che i consumatori abbiano tutti ricevuto i messaggi inviati prima di inviare nuove comunicazioni. Questa caratteristica è ciò che permette di avere l'elevata scalabilità per cui Kafka è noto. I consumatori possono, inoltre, leggere quante volte desiderano i messaggi precedenti, in quanto essi non verranno eliminati dopo la prima lettura.

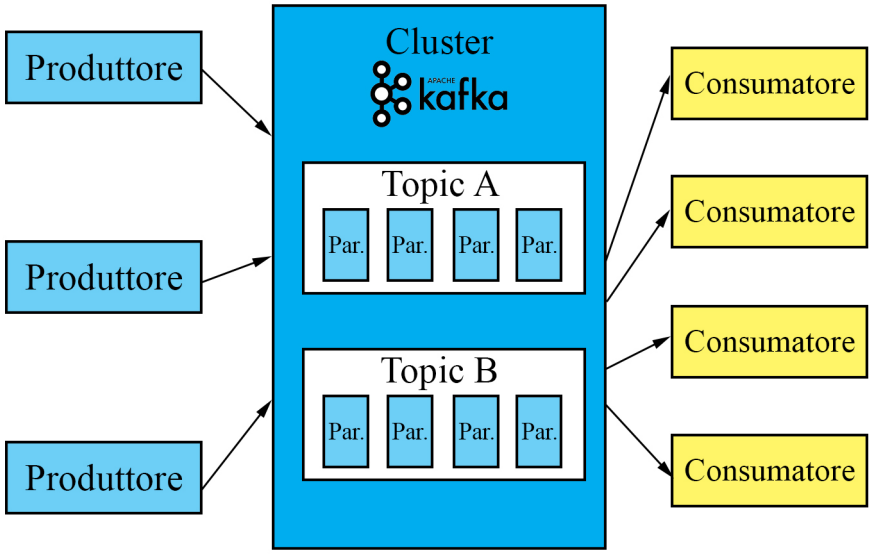


Figura 3.4. Schema architetturale di Apache Kafka

3.5 Neo4j - Database a grafo

Fonte [21]

Neo4j è un database a grafo NOSQL ed opensource, il cui sviluppo è partito nel 2003. Con il termine di database a grafo ci si riferisce a una categoria di software gestori di database i quali integrano in maniera nativa le relazioni fra le informazioni, rendendole parte integrante (insieme ai dati) delle informazioni. I database SQL più conosciuti (come MySql, Oracle, e molti altri), utilizzano un modello per archiviare i dati all'interno di tabelle, ed il calcolo delle relazioni fra i dati stessi viene effettuato al momento delle interrogazioni (utilizzano il comando JOIN), questo però fa' aumentare il tempo necessario per calcolare la risposta all'interrogazione, in quanto le operazioni di JOIN sono molto costose in termini prestazionali. Il beneficio maggiore dell'utilizzo di database a grafo, lo si ottiene con set di dati che presentano molte connessioni fra loro, oltre ad interrogazioni particolarmente complesse. L'accesso a nodi e relazioni avviene a tempo costante, attraversando milioni di connessioni al secondo.

I componenti che compongono il modello di un database a grafo sono solamente due:

- Nodi: essi sono le entità del grafo, le quali possono contenere un numero arbitrario di attributi che prendono il nome di *proprietà*. Per rappresentare

la loro funzione all'interno del grafo, ma anche per poter legare dei metadati, ogni nodo può avere un'etichetta;

- Relazioni: esse sono connessioni dirette fra due entità nodo. Ogni relazione è sempre caratterizzata da un tipo, da una direzione, da un nodo iniziale e uno finale. Esse possono anche avere delle proprietà (come un peso, un costo, ecc...). I nodi possono condividere un numero indefinito di relazioni fra loro, senza che questo pesi sulle prestazioni.

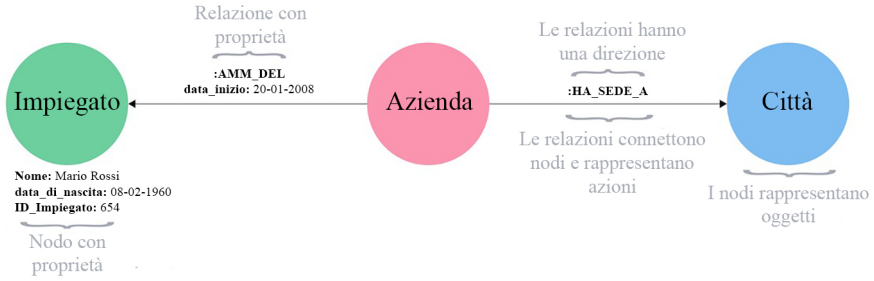


Figura 3.5. Schema di nodi-relazioni all'interno di Neo4j - fonte [21]

Neo4j non utilizza per le query il noto linguaggio SQL (essendo appunto un DBMS NOSQL), ma utilizza un linguaggio simile: Cypher. Tale linguaggio è sempre dichiarativo ma è ottimizzato per i database a grafo. Proprio come i database SQL, anche Neo4j offre driver per i maggiori linguaggi di programmazione, come Java, JavaScript, Python ecc... Esistono molti modi possibili per interagire e utilizzare i dati contenuti in un database Neo4j, la Figura 3.6.

Vediamo ora, attraverso un semplice esempio, come inserire dei dati all'interno di un database Neo4j.

Immaginiamo di avere un database in cui memorizziamo tutte le vendita effettuate presso il nostro negozio. Avremo quindi i seguenti nodi, ognuno caratterizzato da specifiche proprietà:

- **Vendita**:
  - ID;
  - Data;
  - # Articoli;
  - Importo totale;
- **Articolo**:
  - ID;

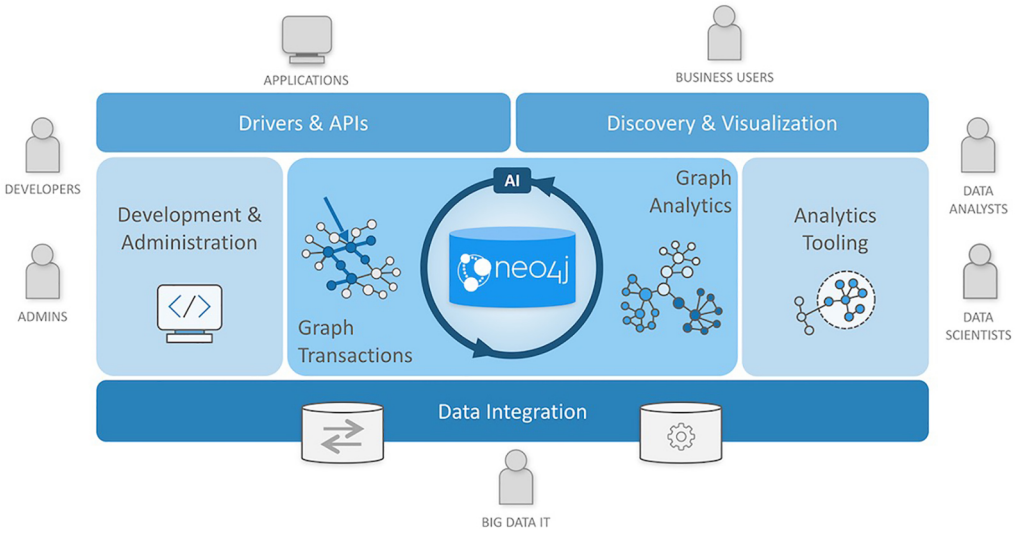


Figura 3.6. Modalità di accesso ai dati in un database Neo4j - fonte [21]

– Nome;
– Prezzo_unitario;

Mentre come relazioni ne avremo di un unico tipo (che avrà come etichetta 'Contiene'), le quali partiranno dal nodo *Vendita* e termineranno sui nodi *Articolo*. Tali relazioni avranno come unica proprietà la quantità di quell'articolo venduta con quella vendita.

Immaginiamo di dover inserire una vendita effettuata in data 15/11/2020 di 3 articoli per un importo totale di euro 25,00. Gli articoli acquistati sono i seguenti:

- ID: 805142 - Pezzi 1 - Confezione 6 bottiglie d'acqua naturale - Prezzo unitario euro 2,00;
- ID: 841396 - Pezzi 1 - Cuffie wireless - Prezzo unitario euro 20,00;
- ID: 887421 - Pezzi 1 - Confezione 2 pile tipo AA - Prezzo unitario euro 3,00

Per prima cosa dovremo inserire gli articoli all'interno del nostro database. Per effettuare questa operazione utilizzeremo le query Cypher riportate nel Listato 3.4.

Una volta inseriti gli articoli, possiamo inserire la vendita con la query Cypher riportata nel Listato 3.5.

Ora rimangono solamente le relazioni da inserire, in modo da legare gli articoli alla vendita. Per fare tale operazione sono sufficienti le query riportate nel Listato 3.6.

Possiamo vedere il risultato finale chiedendo a Neo4j di mostrarci il grafo dei dati in esso contenuti. Tale risultato è mostrato nella Figura 3.7.

Listato 3.4. Query d'inserimento degli articoli all'interno del database Neo4j

```
CREATE (:Articolo { ID: '805142', Nome: 'Confezione 6 bottiglie d\'acqua
↪ naturale', Prezzo_unitario:'2.00' });
CREATE (:Articolo { ID: '841396', Nome: 'Cuffie wireless', Prezzo_unitario:'
↪ 20.00' });
CREATE (:Articolo { ID: '887421', Nome: 'Confezione 2 pile tipo AA',
↪ Prezzo_unitario:'3.00' });
```

Listato 3.5. Query d'inserimento della vendita all'interno del database Neo4j

```
CREATE (:Vendita { ID: '2020111501', Data: '15/11/2020', Num_Articoli: '3',
↪ Importo_totale: '25.00'});
```

Listato 3.6. Query d'inserimento delle relazioni Vendita - Articolo all'interno del database Neo4j

```
MATCH (v:Vendita),(a:Articolo)
WHERE v.ID = '2020111501' AND a.ID = '805142'
CREATE (v)-[r:Contiene { Pezzi: '1' }]->(a);

MATCH (v:Vendita),(a:Articolo)
WHERE v.ID = '2020111501' AND a.ID = '841396'
CREATE (v)-[r:Contiene { Pezzi: '1' }]->(a);

MATCH (v:Vendita),(a:Articolo)
WHERE v.ID = '2020111501' AND a.ID = '887421'
CREATE (v)-[r:Contiene { Pezzi: '1' }]->(a);
```

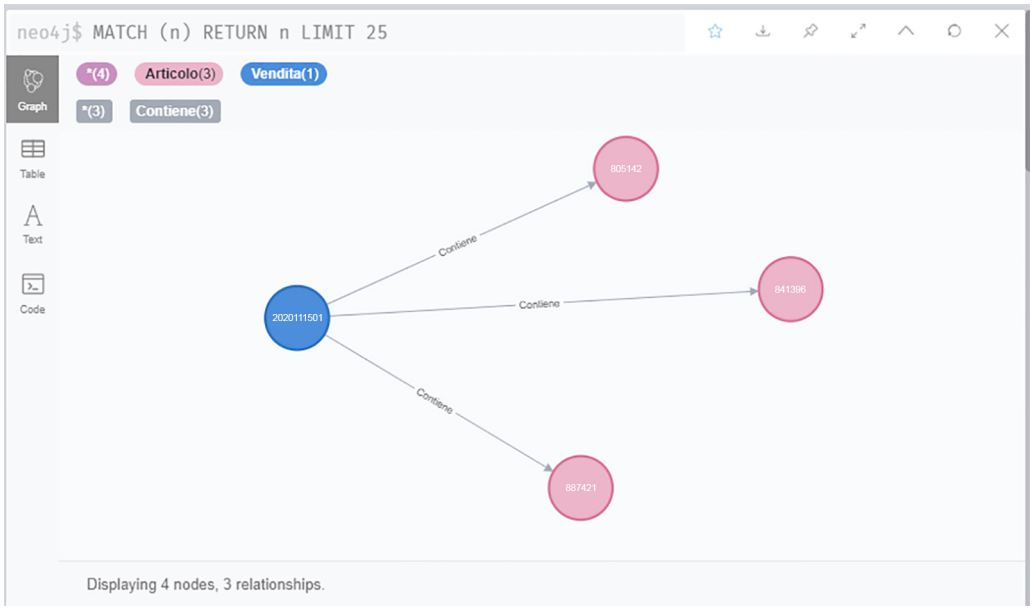


Figura 3.7. Grafo risultante dopo le inserzioni effettuate

Capitolo 4

Obiettivi

Andremo qui a definire quali saranno gli obiettivi finali del nostro lavoro di tesi, illustrando inoltre le metodologie adottate per ottenere tali risultati.

4.1 Progettazione Security Controller v.2

Nella prima parte di questo capitolo introdurremo innanzitutto le motivazioni per cui è necessario implementare una seconda versione del Security Controller, per poi illustrare le fasi di progettazione del controller che hanno preceduto la fase implementativa che affronteremo a partire dal prossimo capitolo. Come è già stato illustrato nel corso dell'introduzione di questa tesi, ormai anche nel mondo delle reti informatiche la virtualizzazione sta diventando sempre più presente, rendendo molto utilizzata la tecnologia NFV illustrata nel capitolo 2.

Grazie allo sfruttamento di queste nuove tecniche, è possibile affidare a fornitori terzi la gestione della sicurezza della propria rete, fattore che permette di diminuire i costi soprattutto per aziende medio/piccole.

Questo però sta a significare che si perde il significato di avere un "perimetro" della rete, in quanto essa si estenderà anche in reti di altri proprietari (il fornitore terzo). Per questo motivo non è più immaginabile pensare di effettuare sicurezza solo sul perimetro, in quanto esso non è più identificabile.

Bisogna anche tener conto che il mondo in cui lavoriamo è un mondo in continua evoluzione, sia per quanto riguarda aspetti positivi ma anche per aspetti negativi; questo significa che anche i vari attacchi informatici sono in continua evoluzione e, con il tempo, diventano sempre più sofisticati.

Ad oggi la sicurezza all'interno delle reti viene svolta mediante diversi tool eterogenei, i quali svolgono compiti spesso individuali e che solo grazie ad intervento umano si riescono a combinare fra loro i vari risultati ed ottenere il livello di sicurezza richiesto. Il livello di reattività del sistema ad un attacco, né risulta quindi

compromesso. Per questo motivo sono necessari dei tool che siano in grado di comunicare fra loro senza intervento umano, in modo da reagire ad eventuali attacchi in maniera automatica, seguendo regole che sono state impostate a propri da umani, notificando poi le azioni intraprese.

La prima versione del Security Controller sviluppata non offre tutta una serie di queste caratteristiche ormai richieste dal mercato, ma permette solamente di avere in output una configurazione ottimale dei firewall all'interno di una rete virtuale, e delle rispettive configurazione che sono necessarie su di essi, ma non reagisce ad eventi che possono accadere all'interno della rete, né è possibile configurare le azioni dall'esterno, ma esse sono programmate all'interno del codice dell'applicativo, quindi non è possibile la loro modifica senza una ricompilazione del codice.

Con questa base di partenza si è iniziata a progettare la seconda versione del Security Controller, il cui flusso di lavoro logico è rappresentato nella Figura 4.1

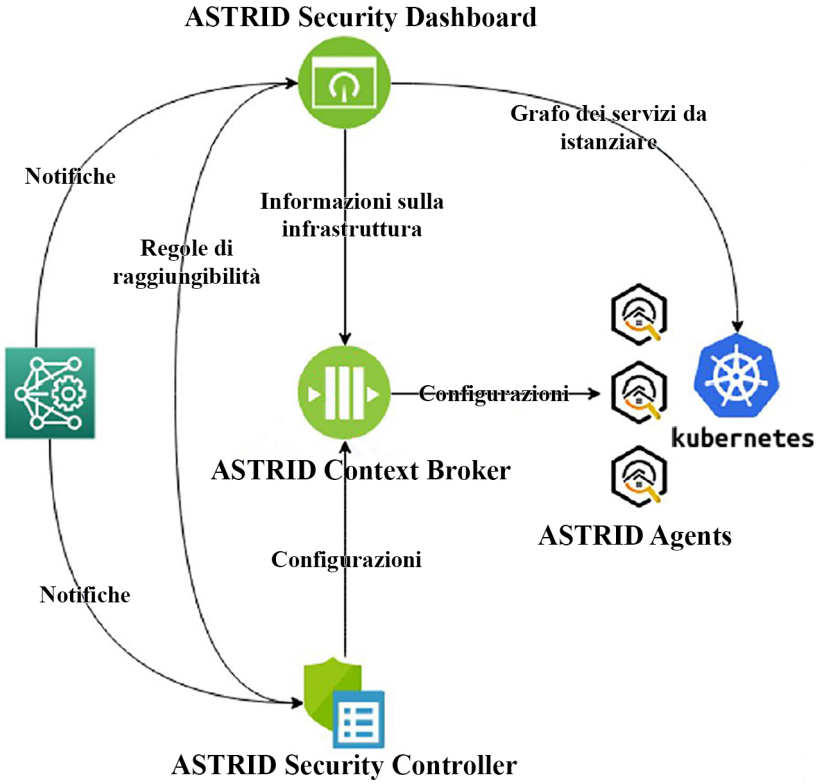


Figura 4.1. Schema logico dell'architettura di ASTRID Security Controller v.2 - fonte: [19]

I componenti che fanno parte dell'architettura ASTRID in questa seconda versione sono i seguenti:

- ASTRID Security Controller: Componente che svilupperemo nel corso di questa tesi. I suoi principali compiti sono i seguenti:
  - Ricevere le politiche di raggiungibilità e fornire la configurazione ottimale dei firewall per garantire tali politiche, oltre alla configurazione di ognuno di essi;
  - Reagire agli eventi che gli vengono notificati con regole che sono state fornite a priori al Security Controller, ma che possono essere modificate nel corso del tempo in maniera semplice, senza mettere mano al codice del componente;
- ASTRID Context Broker: Componente cuore del sistema ASTRID che mantiene tutte le informazioni necessarie al funzionamento dell'intero sistema, e permette lo scambio di esse fra i vari componenti;
- ASTRID Security Dashboard: Interfaccia web da cui avviene la comunicazione sistema ASTRID-umano. Da questa interfaccia il gestore della rete istanzia i vari servizi virtuali, configura le regole di raggiungibilità e verifica gli avvenimenti che sono accadute nel sistema;
- ASTRID Agents: applicativi che vengono istanziati sui vari servizi di rete e che monitorano il loro funzionamento e svolgono le azioni che l'ASTRID Security Controller ordina di effettuare sugli end-system (sono coloro che attuano quindi fisicamente le politiche che vengono configurate dal gestore della rete). I comandi che vengono impartiti dal Security Controller non giungono direttamente agli ASTRID Agents, ma essi vengono inviati all'ASTRID Context Broker, il quale si occupa anche di consegnarli a destinazione;
- L'orchestratore dell'infrastruttura, il quale si occupa di notificare i vari eventi che accadono all'interno della rete all'ASTRID Security Controller ed anche all'ASTRID Security Dashboard;
- Orchestratore di container: nell'Immagine 4.1 è rappresentato Kubernetes. Esso, oltre ai compiti che svolge in quanto orchestratore, riceve dall'ASTRID Security Dashboard le informazioni sui vari servizi che devono essere istanziati all'interno dell'infrastruttura.

Il gestore dell'infrastruttura carica sull'ASTRID Security Dashboard il grafo dell'infrastruttura di rete che vuole realizzare (può essere richiesto, ad esempio, di creare una rete come quella utilizzata nell'esempio illustrato nella prima sezione di questo capitolo, Figura 5.2), per fare questo effettua l'upload del corrispondente file XML che rappresenta la rete da realizzare (Listato 5.3 prendendo sempre in esempio la rete già sfruttata precedentemente). A seguito di questo, deve successivamente decidere come il sistema ASTRID si deve comportare; attualmente sono disponibili 3 politiche possibili:

- *Automatic Firewalling*: questa è la politica già implementata nella prima versione del Security Controller. Il gestore dell'infrastruttura, con questa politica, richiede ad ASTRID il calcolo ottimale della configurazione dei firewall (come visto nell'esempio svolto nella prima sezione di questo capitolo, ma con una differenza. Qui non viene più utilizzata l'API REST e il software Postman per fornire l'infrastruttura, ma la si fornisce da una comoda interfaccia grafica che è l'ASTRID Security Dashboard); è chiaro che il gestore dell'infrastruttura deve fornire anche le regole di raggiungibilità per poter permettere ad ASTRID di svolgere il suo lavoro;

- *Machine Learning Detection*: attivando questa politica si attiva un modulo di Machine Learning il quale, in una prima fase, esso ha il compito di monitorare il funzionamento della rete per apprenderne i meccanismi di funzionamento che vengono sfruttati nella seconda fase per rilevare eventuali attacchi in corso. Nella seconda fase, quindi, che corrisponde alla fase attiva di questo modulo, nel caso in cui venga rilevato del traffico anomalo all'interno dell'infrastruttura, esso genererà le notifiche necessarie per informare il sistema ASTRID (e quindi il modulo di Security Controller), che dovrà agire per bloccare tale minaccia;

- *Network Analytic Toolkit*: politica con la quale si individuano anomalie in modelli di traffico periodici e si utilizza il sistema ASTRID per effettuare verifiche più approfondite in caso di sospetti.

Una volta che il gestore dell'infrastruttura ha effettuato l'upload del grafo da implementare, la Dashboard utilizza le API dell'orchestratore di container per richiedere di istanziare i vari servizi. Fatto ciò l'ASTRID Security Dashboard fornisce all'ASTRID Security Controller l'evento di inizializzazione in base alla politica scelta dal gestore del servizio (ad esempio, nella politica Automatic Firewalling, l'evento di inizializzazione inviato al Security Controller sono le politiche di raggiungibilità). In questo modo l'ASTRID Security Controller inizia ad operare per realizzare la politica di sicurezza richiesta.

Le azioni che vengono svolto sono diverse a seconda del tipo di politica, ma in linea generale possiamo affermare che viene eseguito il calcolo della configurazione ottimale dei firewall, oltre all'attivazione degli ASTRID Security Agent, i quali sono fondamentali per l'attuazione delle politiche, oltre che per il monitoraggio degli avvenimenti e per la notifica degli stessi. Proprio grazie agli ASTRID Security Agent, al verificarsi di un evento all'interno dell'infrastruttura di rete, avviene la generazione di una notifica, la quale viene inviata sia all'ASTRID Security Dashboard, sia all'ASTRID Security Controller, il quale quest'ultimo si preoccupa di realizzare l'azione pre-configurata al verificarsi di tale evento.

Un possibile evento non dannoso che può verificarsi è l'aggiunta o la rimozione di un'istanza di un servizio virtuale (teniamo presente che uno dei punti di forza della tecnologia NFV è la scalabilità, ovvero se cresce il carico di lavoro su un servizio

virtuale istanziate, è possibile istanziare una seconda istanza per bilanciare tale carico). Anche tale evento viene notificato al Security Controller, il quale informerà l'ASTRID Context Broker, il quale gli fornirà il grafo della rete aggiornato, con cui il Security Controller calcolerà la nuova allocazione dei firewall per garantire le politiche di raggiungibilità configurate.

Se il gestore dell'infrastruttura richiede una politica "Machine Learning Detection", gli ASTRID Security Agent vengono configurati in modo che inviino i dati rilevati al modulo di Machine Learning, il quale li utilizza sia per la fase d'apprendimento, ma anche per la fase attiva, nella quale se rileva qualche comportamento sospetto emette una notifica destinata sia all'ASTRID Security Dashboard, sia all'ASTRID Security Controller.

Capitolo 5

Design della soluzione

In questo capitolo descriveremo le fasi implementative dello sviluppo del modulo ASTRID Security Controller versione 2. Prima di illustrare il lavoro svolto, nella prima parte, analizzeremo il codice di partenza che è stato fornito, per individuare le criticità da migliorare al fine di sviluppare una soluzione in linea con gli obbiettivi che sono stati illustrati nel Capitolo 4.1.

5.1 Security Controller v.1

Fonte[3] [18]

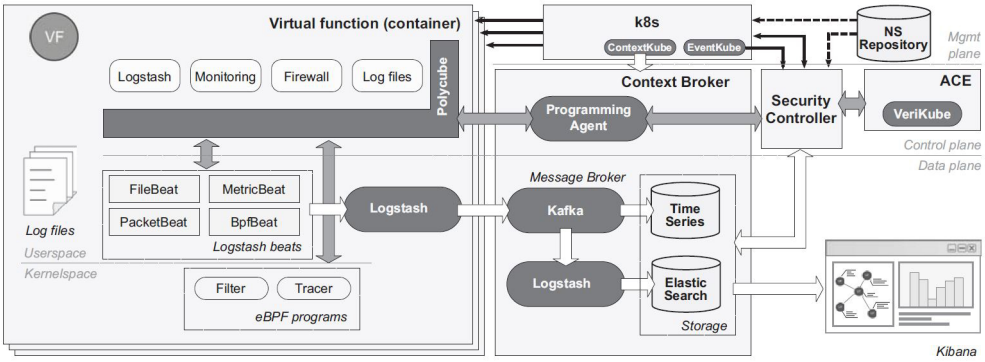


Figura 5.1. Architettura del framework ASTRID - fonte: [3]

La Figura 5.1 mostra lo schema dell'architettura di Astrid da cui si evince che è necessaria una fitta rete di comunicazione fra i vari componenti del sistema. In particolare, il Security Controller ha necessità di comunicare con diversi altri componenti, uno fra tutti è l'orchestratore del servizio (che nello schema è rappresentato

da Kubernetes[1], indicato con la sigla k8s, ma può essere anche di altro tipo, come ad esempio Docker Swarm o Apache Mesos).

Questo componente è integrato nell'architettura ASTRID e per tale architettura è progettato, ma può essere integrato anche in qualsiasi altro progetto, in quanto esso è un'applicazione per Web Server nonché un REST controller, quindi è utilizzabile mediante interfacciamento con API REST.

Nella sua versione iniziale, esso espone 3 API REST, che sono elencate nella Tabella 5.1.

Il Security Controller riceve dall'orchestratore tutta una serie di informazioni preziose sui vari servizi attualmente attivi nella rete, corredati dagli indirizzi IP e dalle rispettive porte.

Uno dei compiti principali del Security Controller è fornire la configurazione ottimale dei firewall e le rispettive policy da applicare, all'interno della rete virtualizzata. Per svolgere questo compito, il Security Controller si interfaccia con un modulo esterno che prende il nome di VereKube o Verefoo (noi utilizzeremo questo secondo termine per riferirsi a tale componente), il quale utilizza la teoria del MaxSMT (Maximum Satisfiability Modulo Theories), che permette di avere 2 caratteristiche di primaria importanza:

- Garanzia formale sulla correttezza della soluzione;
- Ottimalità della soluzione trovata, minimizzando il numero di policy che devono essere configurate in ogni firewall;

Affinchè il Security Controller possa svolgere il proprio compito, è necessario fonirgli una descrizione completa delle politiche di raggiungibilità che si vogliono configurare, ovvero esprimere con quali servizi, ognuno di essi, può comunicare o meno. Per chiarire meglio questo concetto possiamo utilizzare un esempio.

Ipotizziamo di avere necessità di istanziare i seguenti servizi all'interno della nostra rete:

- Un web server Apache Tomcat pubblico raggiungibile da internet;
- 2 istanze di database Neo4j;
- Un web server Ngix.

La nostra rete d'esempio è rappresentata nella Figura 5.2

Oltre a fornire il grafo della nostra rete al Security Controller, è necessario anche fornirgli l'elenco delle policy che devono essere utilizzate, ovvero definire le

[1]Kubernetes è uno dei più famosi orchestratori di container, facente parte della cosiddetta categoria di software "DevOps", ovvero strumenti di gestione dell'infrastruttura, chiamati motori di orchestrazione dei container

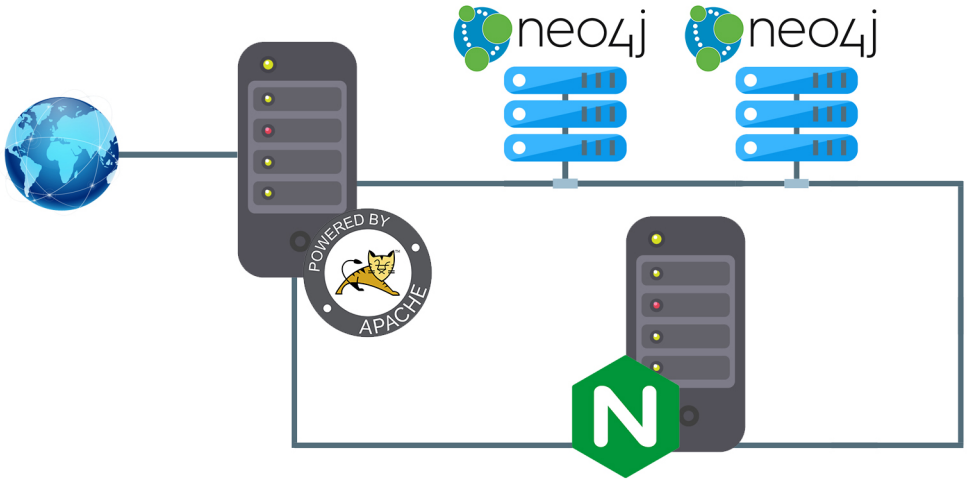


Figura 5.2. Schema logico dei servizi virtuali nella rete d'esempio

regole che permettono ad un servizio di comunicare con un altro e viceversa. Nel nostro esempio, abbiamo il web server Apache che può comunicare con tutti gli altri server, e può comunicare anche con il mondo esterno (internet), mentre tutti gli altri servizi non possono avere alcuna comunicazione se non con il servizio omologo o con il web server Apache. L'unico servizio accessibile dal mondo esterno è il web server Apache. L'orchestratore sarà quindi incaricato di generare il grafo arricchito delle policy che gli sono state definite.

È importante sottolineare il fatto che ogni servizio di rete è istanziato in un POD a se' stante, quindi sarà raggiungibile ad un determinato indirizzo IP e ad una specifica porta di servizio (il tutto è illustrato nella Figura 5.3).

Nello figura 5.3 possiamo anche osservare tutte le possibili comunicazioni nella rete d'esempio; le comunicazioni consentite dalle nostre policy sono raffigurate in nero, mentre le comunicazioni che devono essere negate sono raffigurate in rosso chiaro.

Teniamo presente che l'azione di default impostata nei firewall che devono essere configurati è quella di, in mancanza di regola specifica, bloccare il traffico.

Le policy sono fornite al Security Controller tramite API REST (raggiungibile all'endpoint registerPolicy). Nel Listato 5.1 vediamo il formato corretto con cui queste regole di raggiungibilità vanno fornite al Security Controller, mentre nel Listato 5.2 possiamo vedere le regole di raggiungibilità del caso in esame.

Nel nostro esempio abbiamo fornito le policy rappresentate dal Listato 5.2 tramite l'utilizzo del software Postman, invocando l'API corretta.

La risposta che viene fornita dal Security Controller, se tutto ha avuto successo

Listato 5.1. Scheletro di una regola di raggiungibilità

```
kind: Graph
metadata:
    name: [nome del grafo di rete]
spec:
    policies:
    - from: [nome del servizio mittente]
    to: [nome del servizio destinazione]
    action: [azione da eseguire sul traffico]
```

Listato 5.2. Regole di raggiungibilità configurate

```
kind: Graph
metadata:
    name: Ex_Network
spec:
    policies:
    - from: Apache
      to: Neo4j_1
      action: forward
    - from: Apache
      to: Neo4j_2
      action: forward
    - from: Apache
      to: Ngix
      action: forward
    - from: Ngix
      to: Apache
      action: forward
    - from: Neo4j_2
      to: Apache
      action: forward
    - from: Neo4j_1
      to: Apache
      action: forward
    - from: Neo4j_2
      to: Neo4j_1
      action: forward
    - from: Neo4j_1
      to: Neo4j_2
      action: forward
```

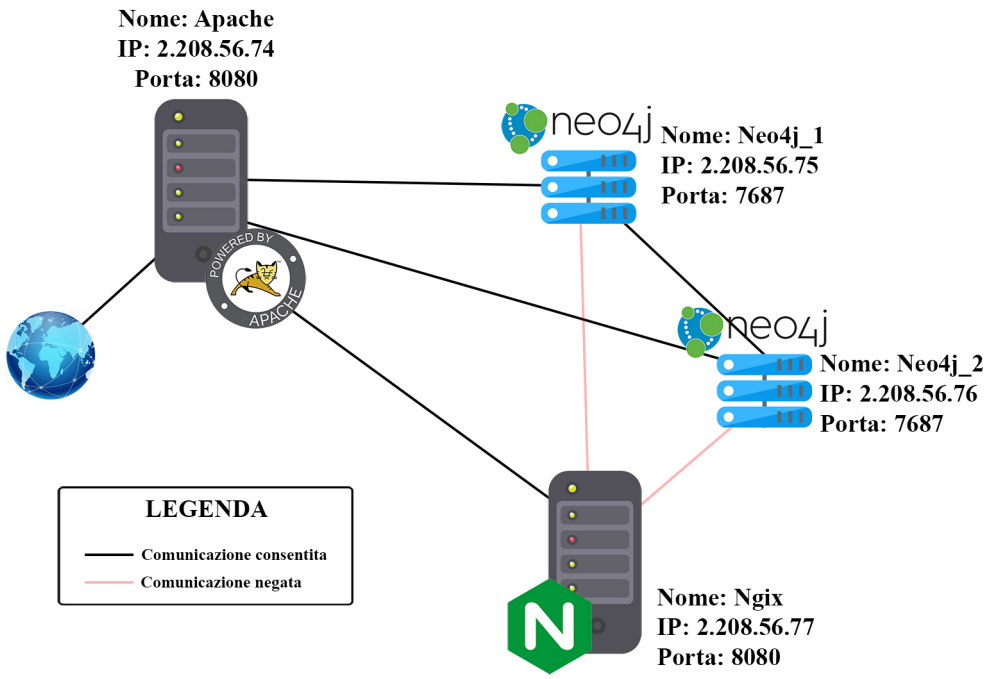


Figura 5.3. Schema logico dei servizi virtuali nella rete d'esempio arricchito dall'orchestratore del servizio

è HTTP Status 200 OK, la quale contiene un messaggio nel corpo che conferma la correttezza dell'operazione (la risposta è raffigurata nella Figura 5.4).

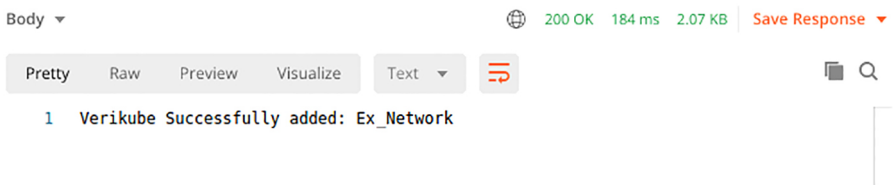


Figura 5.4. Risposta fornita da Verefoo

Il grafo della rete che viene fornito a Verefoo in formato XML, nel caso in esame, è rappresentato nel Listato 5.3.

È importante notare che, il primo passo da effettuare è il fornire la policy che Verefoo deve adottare, e solo dopo fornire il grafo della rete, altrimenti esso verrà rifiutato in quanto non esistono delle politiche di raggiungibilità configurate per tale grafo. Una volta fornite le policy e il grafo della rete, il Security Controller fornisce in output la configurazione ottimale dei firewall nella rete, che garantisce le politiche di raggiungibilità da noi richieste (la risposta completa è disponibile

Listato 5.3. File XML che rappresenta il grafo della rete in esame

```
<InfrastructureInfo>
    <Metadata name="Ex_Network" lastUpdate="2020-10-21T16:09:37.367463224Z"/>
    <Spec>
        <Node ip="192.168.122.171"/>
        <Node ip="192.168.122.93"/>
        <Service name="Apache">
            <Port internal="8080" protocol="TCP" exposed="32643"/>
            <AmbassadorPort internal="9000" Protocol="TCP" exposed="31009"/>
            <Instance ip="2.208.56.74" uid="apache-697b84bd87-b4rkz"/>
        </Service>
        <Service name="Neo4j_1">
            <Port internal="7687" protocol="TCP" exposed="31267"/>
            <AmbassadorPort internal="9000" Protocol="TCP" exposed="30627"/>
            <Instance ip="2.208.56.75" uid="neo4j-55c9447f74-wn5c4"/>
        </Service>
        <Service name="Neo4j_2">
            <Port internal="7687" protocol="TCP" exposed="32192"/>
            <AmbassadorPort internal="9000" Protocol="TCP" exposed="31087"/>
            <Instance ip="2.208.56.76" uid="neo4j-84f7f9f55c-8f7zj"/>
        </Service>
        <Service name="Ngix">
            <Port internal="8080" protocol="TCP" exposed="32964"/>
            <AmbassadorPort internal="9000" Protocol="TCP" exposed="31348"/>
            <Instance ip="2.208.56.77" uid="ngix-v4y83n462c-c2d9l"/>
        </Service>
    </Spec>
</InfrastructureInfo>
```

nell'Appendice tecnica, Figura A.1).

Dalla risposta fornita, possiamo osservare che Verekube ha istanziato 4 firewall per garantire le policy di raggiungibilità richieste. Nella figura 5.5 è illustrato lo schema logico della rete d'esempio con le istanze dei firewall allocate da Verekube.

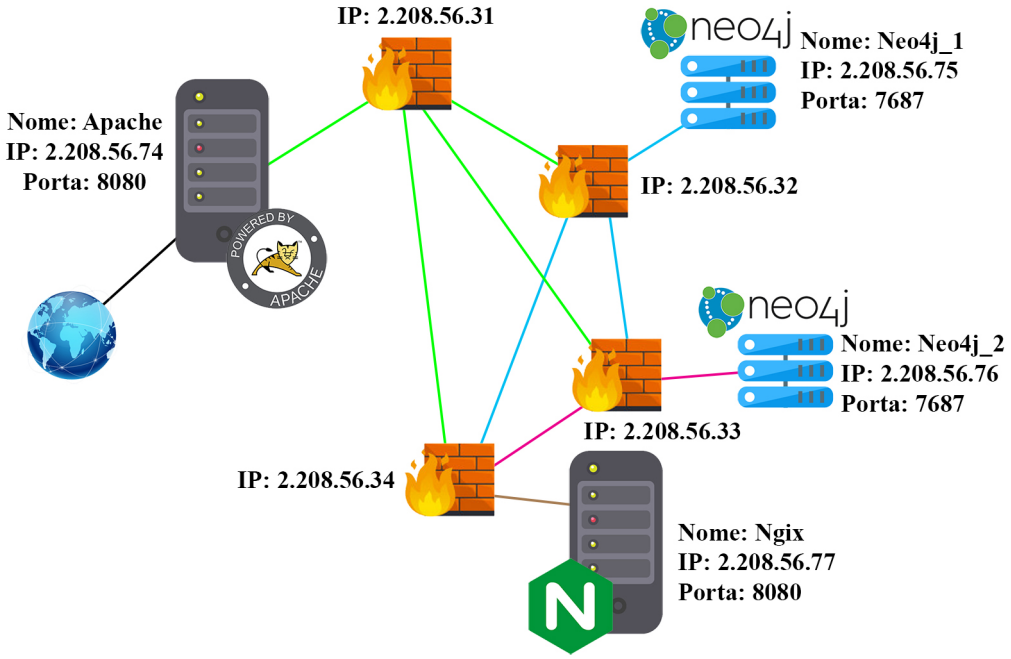


Figura 5.5. Schema logico della rete d'esempio con le istanze dei firewall allocate da Verefoo

Il modulo Verefoo, oltre a fornire l'allocazione ottimale dei firewall, fornisce anche la configurazione per ogni istanza di firewall che garantisce le politiche da noi imposte. Nelle Tabella 5.2, vengono quindi illustrate le regole impostate da Verefoo sui firewall con IP 2.208.56.31, 2.208.56.32, 2.208.56.33, mentre nella Tabella 5.3 sono illustrate le regole impostate sul firewall con IP 2.208.56.34.

5.2 Analisi del punto di partenza

Il codice da cui siamo partiti è ovviamente la prima versione dell'ASTRID Security Controller, nella quale era già stato integrato l'utilizzo del framework Spring Boot illustrato nella sezione 3.2.

Come già affermato, la prima versione del Security Controller non era integrata in alcuna architettura, ovvero non era inserito all'interno di alcuno schema comunicativo; per questo motivo possiamo solamente fare un'analisi statica del codice interno del Security Controller.

Lo schema logico di funzionamento è illustrato nella Figura 5.6, da cui possiamo vedere che erano implementati 3 metodi, i quali gestivano le 3 API REST riassunte nella Tabella 5.1.

Quando il Security Controller riceveva una richiesta REST, grazie al framework Spring, essa veniva gestita dal metodo corrispondente all'endpoint che aveva ricevuto la richiesta; se, ad esempio, si riceveva una richiesta sull'endpoint */registerPolicy*, il metodo che gestiva la richiesta era ovviamente il metodo omonimo, ovvero il metodo registerPolicy; la medesima situazione si aveva con i restanti endpoint possibili.

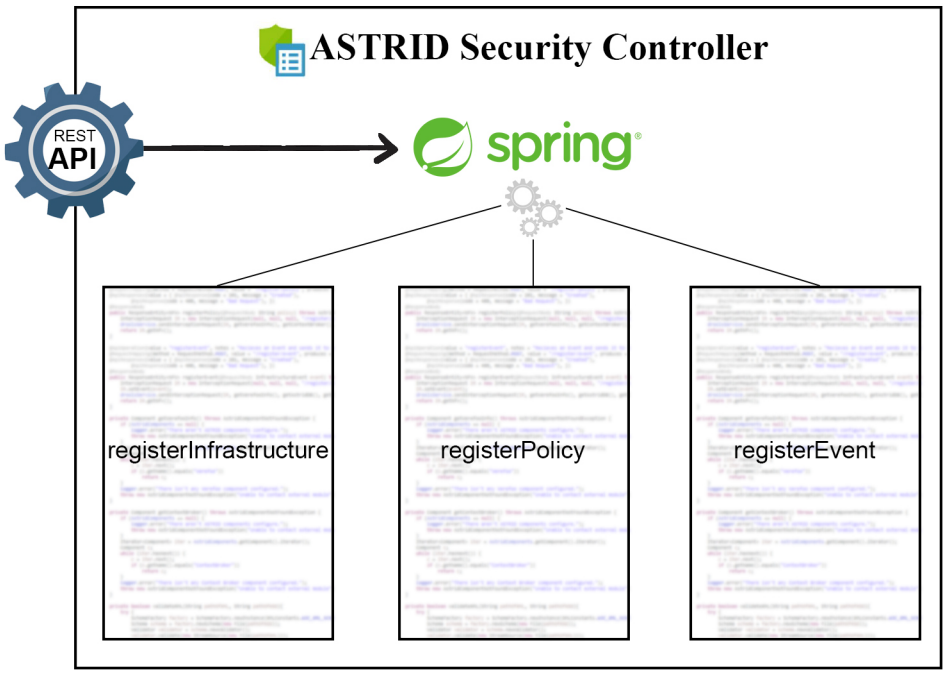


Figura 5.6. Schema logico dell'ASTRID Security Controller v.1

Analizziamo ora i metodi presenti, partendo da registerInfrastructure. Ricordiamo che il compito che deve svolgere tale API è quello di ricevere le informazioni sull'infrastruttura della rete e di inviarle a Verefoo; una volta fatto questo, deve attendere la risposta e fornirla in output.

Nel listato seguente possiamo vedere il codice del metodo:

```
1 @ApiOperation(value = "registerInfrastructure", notes = "Recieves Infrastructure
  ↪ info and sends it to Verikube. Waits for result and sends it back", response =
  ↪ NFV.class)
2 @RequestMapping(method = RequestMethod.POST, value = "/register/insfrastructure",
  ↪  produces = "application/xml", consumes = "application/xml")
3 @ApiResponses(value = { @ApiResponse(code = 201, message = "Created"),
4 @ApiResponse(code = 400, message = "Bad Request"), })
5 @ResponseBody
```

```
6  public NFV registerInfrastructure(@ApiParam(value = "Infrastructure Info",
   ↪ required = true) @RequestBody InfrastructureInfo info) throws
   ↪ ResourceNotFoundException {
7
8      if (info == null || info.getMetadata() == null || info.getMetadata().getName
   ↪ () == null) {
9          System.out.println("++++++++++ registerInfrastructure Empty body");
10         return null;
11     }
12
13     HttpHeaders headers = new HttpHeaders();
14     headers.setContentType(MediaType.APPLICATION_XML);
15
16     System.out.println("++++++++++ registerInfrastructure Controller Obtained
   ↪ Infrastructure info for: " + info.getMetadata().getName());
17     RestTemplate restTemplate = new RestTemplate();
18     Jaxb2RootElementHttpMessageConverter converter = new
   ↪ Jaxb2RootElementHttpMessageConverter();
19     converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
20     restTemplate.setMessageConverters(Arrays.asList(converter, new
   ↪ StringHttpMessageConverter()));
21     // Data attached to the request.
22     HttpEntity<InfrastructureInfo> requestBody = new HttpEntity<
   ↪ InfrastructureInfo>(info, headers);
23
24     System.out.println("++++++++++ registerInfrastructure Controller Sending
   ↪ Infrastructure info to Verekube");
25     // Send request with POST method.
26     NFV result=null;
27     try {
28         result = restTemplate.postForObject(URL_DC, requestBody, NFV.class);
29     } catch (HttpServerErrorException | HttpClientErrorException ex) {
30         System.out.println("++++++++++ registerInfrastructure Controller  Policy
   ↪ is not defined for this graph");
31         throw new ResourceNotFoundException("Policy is not defined for this graph
   ↪ : "+ex.getMessage() );
32     }
33
34     return result;
35     }
```

Listato 5.4. Codice del metodo registerInfrastructure

Le linee di codice dalla 1 alla 5 rappresentano le annotazioni utilizzate dai vari framework/tool:

- Linguaggio documentale Swagger[2]:
  - L'annotazione `@ApiOperation` è utilizzata per descrivere l'azione svolta dall'API REST (linea 1), oltre a descrivere le possibili risposte che essa può generare (`@ApiResponse`, riga 3 e 4);
  - L'annotazione `@ApiParam`(linea 6) è utilizzata per descrivere nella documentazione l'oggetto che il metodo si attende come parametro;

[2]Swagger è un tool opensource per la creazione automatica della documentazione di API REST, ma anche per i test delle stesse

- Spring Framework:
  - L'annotazione `@RequestMapping` viene utilizzata per associare le richieste che giungono con il metodo specificato (`RequestMethod.POST`, linea 2), e con il corretto endpoint (`value="/register/ insfrastructure "`, linea 2), al metodo che ci si accinge a creare. Viene inoltre anche specificato che tipo di dati il metodo si aspetta di ricevere dalla richiesta (`consumes = "application/xml"`, linea 2), e quale tipo di dati la risposta generata conterrà come corpo (`produces = "application/xml"`, linea 2);
  - L'Annotazione `@ResponseBody` è utilizzata per segnalare che l'oggetto che il metodo restituisce deve essere serializzato in XML ed inserito come corpo nella risposta;
  - L'Annotazione `@RequestBody` è utilizzata per segnalare che quell'oggetto deve essere deserializzato dal corpo della richiesta (nel nostro codice, l'oggetto che ci aspetta di avere nel corpo della richiesta è di tipo `InfrastructureInfo` , ovvero il grafo della rete).

Nelle righe dalla 8 alla 11 viene semplicemente fatto un controllo per assicurarsi che il metodo abbia ricevuto il parametro atteso e che esso contenga un nome valido.

Nella riga 13 viene creato un oggetto di tipo *HttpHeader*, che altro non è che l'header della richiesta HTTP che verrà poi inviata a Verefoo. Nella riga seguente si inserisce all'interno dell'header appena creto, l'informazione sul tipo di dato che è contenuto nel body; nel caso in esame si invieranno le informazioni sull'infrastruttura di rete in formato XML, quindi viene indicato `MediaType.APPLICATION_XML`.

Per fornire un feedback sull'andamento delle operazioni svolte dal Security Controller, vengono spesso stampati dei messaggi su console. Lo possiamo vedere nelle righe 16, 24 e 30, dove viene segnalato, rispettivamente: la ricezione di un grafo di rete con un certo nome, la conferma dell'avvenuto invio del grafo al modulo esterno Verefoo, e l'eventuale errore che tale modulo può inviarci come risposta.

Nelle successive righe viene istanziato un oggetto di ripo RestTemplate (riga 17) il quale è un REST client offerto dal framework Spring, utilizzato poi per inviare la richiesta a Verefoo; vengono eseguite delle operazioni per impostare nel restTemplate appena creato un convertitore per poi effettuare la deserializzazione dell'oggetto che verrà ricevuto (righe dalla 18 alla 20).

La creazione della richiesta viene effettuato alla riga 22, dove si istanzia un oggetto di tipo *HttpEntity<InfrastructureInfo>* al cui interno viene inserito l'header HTTP creato precedentemente e il corpo, il quale è costituito dall'InfrastructureInfo che è stato ricevuto come parametro.

L'invio della richiesta viene realizzato a riga 28, utilizzando il metodo `postForObject (URL_DC, requestBody, NFV.class)`, il quale genera una richiesta con metodo POST destinata a URL_DC, nel cui corpo è contenuto *requestBody*, e la risposta sarà deserializzata in un oggetto di classe *NFV*. Tale invio è racchiuso all'interno di un blocco try-catch, necessario per poter catturare eventuali eccezioni che si dovessero sollevare.

Si nota che vengono catturate solamente eccezioni di due tipi: *HttpServerErrorException* e *HttpClientErrorException*, a seguito delle quali si stampa un messaggio in console e si lancia un tipo d'eccezione personalizzato: *ResourceNotFoundException* demandando quindi la gestione ad altri metodi.

Nella riga 34 termina il metodo, il quale restituisce semplicemente un oggetto di tipo *NFV*, il quale verrà serializzato da Spring in XML ed inserito nel corpo della risposta che verrà inviata.

Analizziamo ora il codice del metodo registerPolicy, il quale riceve le politiche di raggiungibilità che si vogliono ottenere sulla rete:

```
1  @ApiOperation(value = "registerPolicy", notes = "Recieves Policies as String and
   ↪ sends it to Verikube. ")
2  @RequestMapping(method = RequestMethod.POST, value = "/register/policy", produces
   ↪  = "text/plain", consumes = "text/plain")
3  @ApiResponses(value = { @ApiResponse(code = 201, message = "Created"),
4  @ApiResponse(code = 400, message = "Bad Request"), })
5  @ResponseBody
6  public String registerPolicy(@RequestBody String policy) {
7      HttpHeaders headers = new HttpHeaders();
8      headers.setContentType(MediaType.TEXT_PLAIN);
9
10     System.out.println("++++++++++ registerPolicy Controller Obtained Policy");
11     RestTemplate restTemplate = new RestTemplate();
12
13     // Data attached to the request.
14     HttpEntity<String> requestBody = new HttpEntity<>(policy, headers);
15
16     System.out.println("++++++++++ registerPolicy Controller Sending to Verekube"
   ↪ );
17     // Send request with POST method.
18     ResponseEntity<String> result = restTemplate.postForEntity(URL_POLICY,
   ↪ requestBody, String.class);
19     // Code = 200.
20     if (result.getStatusCode() == HttpStatus.OK) {
21         System.out.println("++++++++++ registerPolicy Controller Success from
   ↪ Verekube: " + result.getBody());
22     }
23     return result.getBody();
24 }
```

Listato 5.5. Codice del metodo registerPolicy

Si nota che il codice è molto simile a quello appena analizzato, quindi evidenzieremo solamente le differenze: il metodo restituisce come risultato una stringa, la quale si riceve come risposta direttamente dal modulo esterno Verefoo, quindi non è necessaria alcuna conversione o mappatura di quanto ricevuto su oggetto Java; questo è il motivo per cui in questo listato non abbiamo le righe dalla 18 alla 20 del Listato 5.4.

Oltre a restituire una stringa, il metodo accetta anche come unico parametro una stringa, che si aspetta di ottenere dal corpo della richiesta.

Notiamo, nella riga 18, l'invio della richiesta a Verefoo, il quale viene effettuato con il metodo `postForEntity(URL_POLICY, requestBody, String.class)` analogo al metodo utilizzato a riga 28 del Listato 5.4.

Analizziamo, infine, il codice del metodo registerEvent, il quale riceve in formato XML gli eventi accaduti all'interno dell'infrastruttura ed il suo compito è di garantire comunque le politiche di raggiungibilità imposte anche a seguito del verificarsi di tali situazioni, riadattando, se necessario, l'allocazione dei firewall o modificandone le configurazioni.

Il codice è il seguente:

```
1  @ApiOperation(value = "registerEvent", notes = "Recieves an Event and sends it to
   ↪  Verikube. ")
2  @RequestMapping(method = RequestMethod.POST, value = "/register/event", produces
   ↪ = "application/xml", consumes = "application/xml")
3  @ApiResponses(value = { @ApiResponse(code = 201, message = "Created"),
4  @ApiResponse(code = 400, message = "Bad Request"), })
5  @ResponseBody
6  public NFV registerEvent(@RequestBody InfrastructureEvent event) throws
   ↪ ResourceNotFoundException {
7      HttpHeaders headers = new HttpHeaders();
8      headers.setContentType(MediaType.APPLICATION_XML);
9
10     System.out.println("++++++++++ registerEvent Controller Obtained Event");
11
12     RestTemplate restTemplate = new RestTemplate();
13     Jaxb2RootElementHttpMessageConverter converter = new
   ↪ Jaxb2RootElementHttpMessageConverter();
14     converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
15     restTemplate.setMessageConverters(Arrays.asList(converter, new
   ↪ StringHttpMessageConverter()));
16     HttpEntity<InfrastructureEvent> requestBody = new HttpEntity<
   ↪ InfrastructureEvent>(event, headers);
17
18     System.out.println("++++++++++ registerEvent Controller Sending Event info to
   ↪  Verekube");
19
20     NFV result=null;
21     try {
22          result = restTemplate.postForObject(URL_EVENT, requestBody, NFV.class);
23     } catch (HttpServerErrorException | HttpClientErrorException ex) {
24         System.out.println("++++++++++ registerInfrastructure Controller  Policy
   ↪ is not defined for this graph " + ex.getResponseBodyAsString());
25         throw new ResourceNotFoundException("Error: "+ex.getResponseBodyAsString
   ↪ () );
26     }
27
28     return result;
29 }
30 }
```

Listato 5.6. Codice del metodo registerEvent

Il codice del metodo in esame è praticamente identico al codice del metodo *registerInfrastructure* del Listato 5.4, quindi non ci soffermiamo maggiormente per evitare di essere ripetitivi e di annoiare il lettore.

Da come si è potuto vedere dall'analisi dei codici dei vari metodi, all'interno di essi è concentrata tutta la logica di controllo e di gestione per elaborare e per rispondere a tale richiesta. Questo significa che non è possibile in alcun modo modificare il comportamento che il Security Controller assume nella gestione della richiesta

REST dall'esterno, ma l'unica possibilità sarebbe quella di andare a modificare fisicamente il codice all'interno del metodo, e ricompilare l'intero modulo. È evidente che questa operazione può però essere compiuta solo da personale tecnico, il quale ha le capacità e le conoscenze per eseguire tali operazioni, ma il nostro obiettivo è quello di creare un qualcosa il cui comportamento in risposta a stimoli esterni, possa essere modificato nel corso del tempo da chiunque, con strumenti semplici.

L'ASTRID Security Controller necessita di una stretta collaborazione con un modulo esterno: Verefoo. Tale modulo fa' comunque sempre parte dell'architettura ASTRID. Esso ha il compito di fornire la configurazione ottimale dei firewall all'interno della rete, in modo da garantire le politiche di raggiungibilità che vengono richieste dal gestore della rete. L'interfacciamento con Verefoo avviene sempre tramite API REST, il che significa che è necessario fornire al Security Controller l'indirizzo IP e la porta (oltre che l'endpoint) su cui è contattabile l'API REST esposta dal modulo Verefoo che fornisce tale informazione. Da come si è visto dai codici dei metodi, tale informazione viene fornira mediante una costante diversa per ogni metodo e che riportiamo qui di seguito:

- Per il metodo *registerInfrastructure*(Listato 5.4), si utilizza la costante URL_DC (riga 28), la quale è così definita: `static final String URL_DC = "http://localhost:8085/verefoo/dc";`
- Per il metodo *registerPolicy*(Listato 6.2), si utilizza la costante URL_POLICY (riga 18), la quale è così definita: `static final String URL_POLICY = "http://localhost:8085/verefoo/graph";`
- Per il metodo *registerEvent*(Listato 5.6), si utilizza la costante URL_EVENT (riga 22), la quale è così definita: `static final String URL_EVENT = "http://localhost:8085/verefoo/podevent";`

Nella prima versione del Security Controller, tale informazione era quindi inserita come costante all'interno del codice sorgente, con tutti i problemi che ne derivano: al variare di anche solo una componente dell'indirizzo di Verefoo o dell'URL dell'endpoint, era necessario modificare tale informazione all'interno del codice del Security Controller e ricompilare il tutto. Anche qui era sempre necessario l'intervento tecnico di personale qualificato, oltre al fatto che veniva meno l'indipendenza del codice dalla macchina su cui esso viene eseguito.

Un ulteriore argomento ancora da affrontare, era la gestione degli errori che potevano accadere durante l'esecuzione del Security Controller, ovvero non era ancora presente una logica di gestione delle eccezioni uniforme in tutto il codice, in modo da garantire comunque la prosecuzione oppure, nel caso di errore grave, la terminazione ordinata dell'operazione o addirittura del modulo stesso.

5.3 Miglioramento dell'indipendenza

Ai giorni nostri non è più pensabile di scivere un codice destinato all'esecuzione su una specifica macchina, in quanto ormai tutto è connesso e si vuole poter avere accesso alle applicazioni da qualunque luogo. E' quindi fondamentale il concetto di portabilità del codice, al fine di renderlo indipendente dall'hardware sottostante.

Per questo motivo, una grossa limitazione dell'ASTRID Security Controller v.1 era senza dubbio la presenza delle costanti a rappresentare gli endpoint da contattare per l'esecuzione delle operazioni.

Per rimediare a questa problematica si è pensato all'introduzione di un file XML di configurazione; all'interno di tale file è necessario configurare tutti i componenti con cui il Security Controller deve comunicare, nella nostra trattazione si tratta dei moduli Verefoo e Context Broker.

Si è quindi creato uno schema adatto a tale scopo e che riportiamo qui di seguito:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
↪ xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="components">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="component" maxOccurs="unbounded" minOccurs="0">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:attribute type="xsd:string" name="name" use="required"/>
                <xsd:attribute type="xsd:string" name="IPAddress" use="required"/
↪ >
                <xsd:attribute type="xsd:integer" name="Port" use="required"/>
                <xsd:attribute type="xsd:string" name="Username" use="optional"/>
                <xsd:attribute type="xsd:string" name="Password" use="optional"/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listato 5.7. Schema Astrid-components.XSD

All'avvio del Security Controller, quindi, esso controllerà all'interno del file Astrid-components.XML la presenza dei vari componenti necessari all'esecuzione delle operazioni e segnalerà la mancanza o comunque le problematiche che dovesse rilevare.

Quando si introduce l'utilizzo di un file XML, però, prima del suo utilizzo è necessario accertarsi che tale file sia valido rispetto allo schema a cui si riferisce (nel nostro caso, rispetto allo schema rappresentato nel Listato 5.7). Ecco quindi che abbiamo all'interno del costruttore che si occupa di istanziare il REST Controller il codice necessario alla validazione del file XML; se tale file risulta valido, esso eseguirà anche il caricamento dei componenti in memoria in modo che possano essere

poi utilizzati dal Security Controller, altrimenti segnalerà le anomalie riscontrate. Rappresentiamo qui di seguito tale codice:

```
1  Components AstridComponents = null;
2  /* verify if the XML file with configuration of ASTRID components is valid*/
3  boolean validation = validateXML("src/main/resources/Astrid-components.xml", "xsd
   ↪ /Astrid-components.xsd");
4  if(validation != true) {
5      logger.info("The XML file with configuration of ASTRID components, it is NOT
   ↪ valid");
6  }else {
7      /* upload the ASTRID component */
8      try {
9          InputStream inStream = new FileInputStream("src/main/resources/Astrid-
   ↪ components.xml");
10         JAXBContext jaxbContext = JAXBContext.newInstance(Components.class);
11         Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
12         AstridComponents = (Components) unmarshaller.unmarshal(inStream);
13     } catch (JAXBException | FileNotFoundException e) {
14         // TODO Auto-generated catch block
15         e.printStackTrace();
16     }
17 }
18
19 [...]
20
21 private boolean validateXML(String pathOfXML, String pathOfXSD){
22     try {
23         SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.
   ↪ W3C_XML_SCHEMA_NS_URI);
24         Schema schema = factory.newSchema(new File(pathOfXSD));
25         Validator validator = schema.newValidator();
26         validator.validate(new StreamSource(new File(pathOfXML)));
27     } catch (SAXException e) {
28         // TODO Auto-generated catch block
29         logger.info("XML file is NOT valid: "+e.getMessage());
30         return false;
31     } catch (IOException e) {
32         // TODO Auto-generated catch block
33         logger.info("Unable to validate Astrid-components.XML: "+e.getMessage());
34         return false;
35     }
36     return true;
37 }
```

Listato 5.8. Codice inserito all'interno del costruttore del REST Controller che si occupa della validazione del file XML e del successivo caricamento dei componenti in memoria

Da come risulta dal codice fornito, nel momento in cui avviene la creazione del REST Controller viene eseguita la validazione del file XML, tramite il metodo `private boolean validateXML(String pathOfXML, String pathOfXSD)`, il quale riceve come parametri il percorso del file XML da validare ed il percorso dello schema. Tale metodo utilizza gli oggetti *SchemaFactory* e *Validator* per eseguire il suo compito, scatenando due eccezioni: la prima *SAXException* viene scatenata quando il file XML non risulta valido rispetto allo schema, mentre la seconda eccezione possibile *IOException* viene scatenata quando l'Astrid Controller non è in grado di accedere ai percorsi forniti.

In entrambe i casi, il metodo restituisce *false*, facendo fallire la validazione inserendo nel file di log la motivazione che ha causato tale fallimento.

Se la validazione ha esito positivo, durante l'esecuzione del costruttore del REST Controller alla riga 9, vediamo che viene eseguito il caricamento del file XML in memoria per la sua lettura, che viene eseguita tramite un oggetto di tipo JAXBContext che permette di eseguire l'unmarshaller[3] del file. Tale operazione è possibile in quanto è stato eseguito precedentemente il binding dal sistema Maven, il quale ha creato la classe di tipo Components a partire dalla lettura dello schema rappresentato nel Listato 5.7; tale classe ospita al suo interno i campi definiti nello schema, oltre ai rispettivi metodi getter e setter:

```
1  / This file was generated by the JavaTM Architecture for XML Binding(JAXB)
   ↪ Reference Implementation, v2.2.11
2  // See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
3
4  [...]
5
6  @XmlAccessorType(XmlAccessType.FIELD)
7  @XmlType(name = "", propOrder = {
8      "component"
9  })
10 @XmlRootElement(name = "components")
11 public class Components {
12
13     protected List<Components.Component> component;
14
15     /**
16      * Gets the value of the component property.
17      *
18      * <p>
19      * This accessor method returns a reference to the live list,
20      **/
21
22 [...]
23
24     @XmlAccessorType(XmlAccessType.FIELD)
25     @XmlType(name = "", propOrder = {
26         "value"
27     })
28     public static class Component {
29
30         @XmlValue
31         protected String value;
32         @XmlAttribute(name = "name", required = true)
33         protected String name;
34         @XmlAttribute(name = "IPAddress", required = true)
35         protected String ipAddress;
36         @XmlAttribute(name = "Port", required = true)
37         protected BigInteger port;
38         @XmlAttribute(name = "Username")
39         protected String username;
40         @XmlAttribute(name = "Password")
41         protected String password;
```

[3]l'operazione di unmarshaller permette di ottenere un oggetto JAVA a partire da una sua rappresentazione in formato XML

```
42
43 [...]
```

Listato 5.9. Classe Components

Un esempio di configurazione del file Astrid-components.xml è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <component name='Verefoo' IPAddress='localhost' Port='8085'/>
    <component name='ContextBroker' IPAddress='localhost' Port='8088'/>
</components>
```

Listato 5.10. esempio del file Astrid-components.xml

Nell'esempio precedente abbiamo configurato il componente Verefoo (raggiungibile all'indirizzo IP *localhost* e alla porta *8085*), ed il componenete Context Broker (raggiungibile all'indirizzo IP *localhost* e alla porta *8088*). Quando si esegue il modulo Security Controller su un'altra macchina, oppure quando gli indirizzi dei componenti esterni subiscono variazione, è sufficiente modificare tale file di configurazione con i nuovi parametri ed eseguire nuovamente il Security Controller, senza mettere mano al codice e senza la necessità di ricompilazioni.

5.4 Introduzione delle regole Drools

Come già più volte affermato, uno degli obiettivi principali della seconda versione del Security Controller era quello di permettere la modifica del suo comportamento a seguito di input ricevuti, dall'esterno del modulo, senza intervento sul codice. Per questo scopo era necessario estrarre la logica di funzionamento del modulo dal codice stesso. A tal fine si è ricorso all'utilizzo delle Drools Rules, la cui spiegazione è stata fornita nel Capitolo 3.3.

Per sfruttare tali regole si è creata una classe di supporto (DroolsService), la quale ha il compito di istanziare il motore delle regole necesario per l'elaborazione delle stesse e una classe di configurazione (DroolsConfiguration), oltre ad un file *.drl* il quale è colui che conterrà le varie regole di funzionamento.

Era poi necessario frammentare le varie operazioni da svolgere in operazioni più semplici, oltre alla necessità di avere una classe di supporto in grado di incapsulare le varie informazioni utili fra i vari metodi di elaborazione. A tale scopo, è stata utilizzata una classe particolare, battezzata *InterceptionRequest*, formata dalle seguenti variabili:

```
1 public class InterceptionRequest {
2     String userId;
3     String providerId;
4     String serviceId;
5     String command;
6     InfrastructureInfo info;
7     String policy;
8     InfrastructureEvent event;
9     ResponseEntity<NFV> nfv;
```

```
10      ResponseEntity<String> result;
11  ...
12  }
```

Listato 5.11. Campi della classe InterceptionRequest

Nella classe *InterceptionRequest*, oltre ai campi illustrati, sono presenti solamente i metodi getter e setter relativi.

Quando giunge una richiesta REST, grazie al framework Spring, essa giunge al metodo di gestione, così come illustrato all'inizio di questo capitolo. Ai metodi di gestione sono state apportate, però, modifiche significative, in quanto è stata estratta tutta la logica di gestione in quanto essa verrà svolta da un metodo successivo. Il compito che è rimasto in carico ai metodi di gestione è quello di creare un oggetto della classe InterceptionRequest, impostare nel campo *command* la stringa a cui quell'endpoint è stato raggiunto (ad esempio, se è il metodo registerPolicy che sta gestendo la richiesta, all'interno della stringa *command* verrà settatto "/register/policy"), inserire all'interno dell'oggetto anche l'informazione che si è ricevuto come corpo nella richiesta (ad esempio, per il metodo registerInfrastructure nel corpo della richiesta avremo il grafo della rete in formato XML, il quale è stato automaticamente mappato su un oggetto di tipo InfrastructureInfo dal framework Spring) e successivamente inserire tale oggetto all'interno del motore delle regole Drools, insieme ai vari componenti necessari per la crretta gestione (ad esempio il modulo Verefoo e/o il Context Broker).

Nei listati seguente possiamo vedere le implementazioni dei vari metodi dopo le nostre modifiche, nell'ordine: *registerPolicy*, *registerInfrastructure* ed *registerEvent*:

```
1  @ApiOperation(value = "registerPolicy", notes = "Recieves Policies as String and
   ↪ sends it to Verikube. ")
2  @RequestMapping(method = RequestMethod.POST, value = "/register/policy", produces
   ↪  = "text/plain", consumes = "text/plain")
3  @ApiResponses(value = { @ApiResponse(code = 201, message = "Created"),
4  @ApiResponse(code = 400, message = "Bad Request"), })
5  @ResponseBody
6  public ResponseEntity<NFV> registerPolicy(@RequestBody String policy) throws
   ↪ AstridComponentNotFoundException, IOException, ResourceNotFoundException {
7      InterceptionRequest IR = new InterceptionRequest(null, null, null, "/register
   ↪ /policy", null, policy);
8      droolsService.sendInterceptionRequest(IR, getVerefooInfo(), getContextBroker
   ↪ (), null);
9      return IR.getNfv();
10 }
11 }
```

Listato 5.12. Implementazione del metodo registerPolicy dopo le modifiche apportate

```
1  @ApiOperation(value = "registerInfrastructure", notes = "Recieves Infrastructure
   ↪ info and sends it to Verikube. Waits for result and sends it back", response =
   ↪ NFV.class)
2  @RequestMapping(method = RequestMethod.POST, value = "/register/insfrastructure",
   ↪  produces = "application/xml", consumes = "application/xml")
3  @ApiResponses(value = { @ApiResponse(code = 201, message = "Created"),
4  @ApiResponse(code = 400, message = "Bad Request"), })
5  @ResponseBody
```

```
6  public ResponseEntity<NFV> registerInfrastructure(
7  @ApiParam(value = "Infrastructure Info", required = true) @RequestBody
   ↪ InfrastructureInfo info) throws ResourceNotFoundException,
   ↪ AstridComponentNotFoundException {
8      InterceptionRequest IR = new InterceptionRequest(null, null, null, "/register
   ↪ /insfrastructure", info, null);
9      droolsService.sendInterceptionRequest(IR, getVerefooInfo(), null, null);
10     return IR.getNfv();
11 }
12 }
```

Listato 5.13. Implementazione del metodo registerInfrastructure dopo le modifiche apportate

```
1  @ApiOperation(value = "registerEvent", notes = "Recieves an Event and sends it to
   ↪  Verikube. ")
2  @RequestMapping(method = RequestMethod.POST, value = "/register/event", produces
   ↪ = "application/xml", consumes = "application/xml")
3  @ApiResponses(value = { @ApiResponse(code = 201, message = "Created"),
4  @ApiResponse(code = 400, message = "Bad Request"), })
5  @ResponseBody
6  public ResponseEntity<NFV> registerEvent(@RequestBody InfrastructureEvent event)
   ↪ throws ResourceNotFoundException, AstridComponentNotFoundException {
7      InterceptionRequest IR = new InterceptionRequest(null, null, null, "/register
   ↪ /event", null, null);
8      IR.setEvent(event);
9      droolsService.sendInterceptionRequest(IR, getVerefooInfo(), getAstridDB(),
   ↪ getContextBroker());
10     return IR.getNfv();
11 }
```

Listato 5.14. Implementazione del metodo registerEvent dopo le modifiche apportate

E' importante notare che, in tutti i metodi ora vengono svolte pressochè le medesime azioni ma con diversi parametri: si crea un oggetto di tipo InterceptionRequest, inserendo all'interno quanto ricevuto dalla richiesta, oltre all'indirizzo relativo a quell'endpoint, e successivamente si inserisce l'oggetto e i componenti utili per la sua gestione nel motore delle regole Drools.

Tutta la logica di gestione della richiesta ricevuta è ora stata inserita all'interno di metodi creati appositamente per tale scopo; tali metodo sono quindi dei metodi di servizio, e prendono effettivamente il nome di: *registerInfrastructureService*, *RegisterPolicyService* e *RegisterEventService*.

Nella Figura 5.7 possiamo vedere lo schema logico dell'architettura del Security Controller dopo le modifiche effettuate.

Il motore delle regole, per scegliere l'azione da svolgere si deve basare sulle regole definite dall'utente; tali regole sono inserite all'interno di un file con estensione *.drl*. Il file che contiene le regole del Security Controller si chiama *interception.drl*, il cui attuale contenuto è rappresentato nel Listato A.2 riportato nell'appendice tecnica.

La prima regola introdotta *Register infrastructure info* è colei che stabilisce le azioni da svolgere quando si riceve una richiesta sull'endpoint "/register/insfrastructure", la seconda regola *Register Policy* viene svolta quando si riceve una richiesta sull'endpoint "/register/policy", mentre la terza regola *Register Event - Add a Instance* e la quarta *Register Event - Remove a Instance* vengono svolte quando si

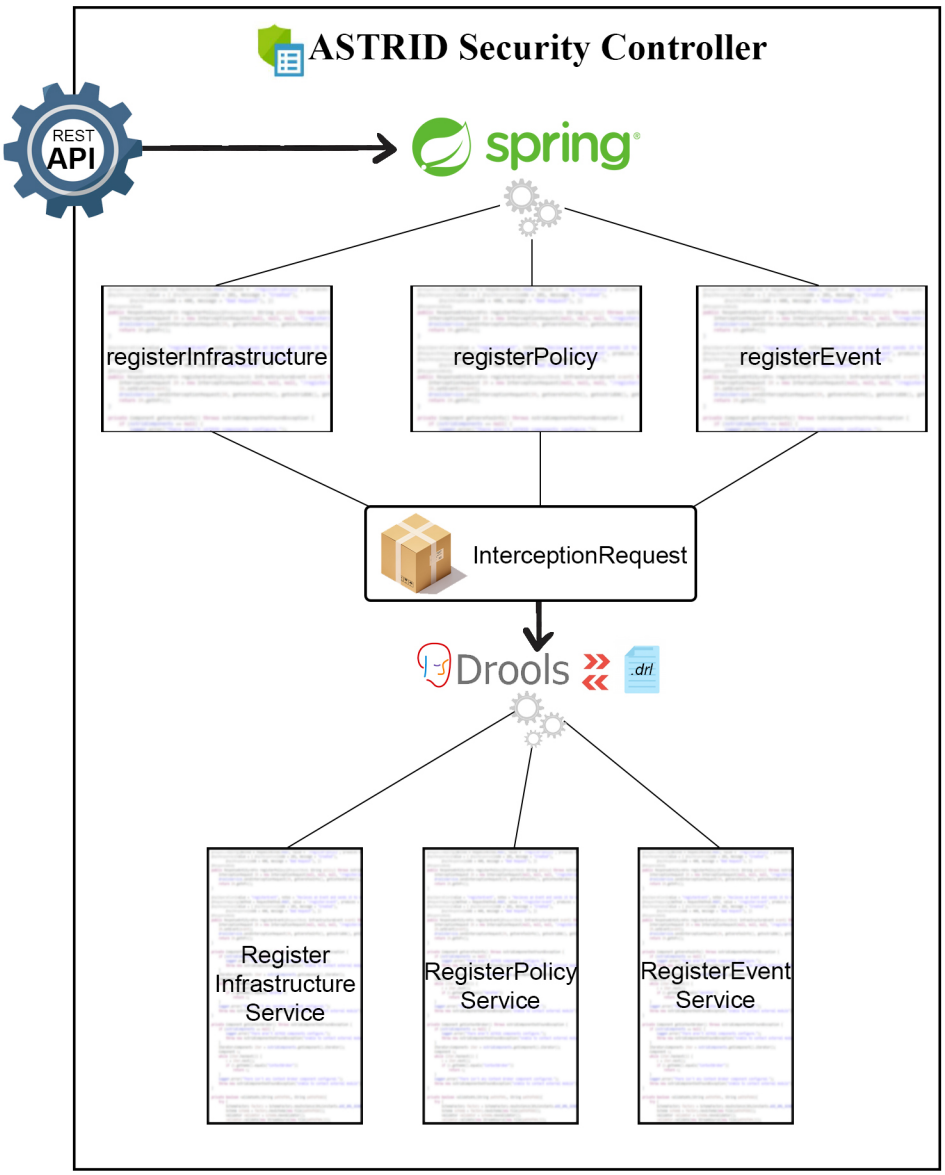


Figura 5.7. Schema logico dell'ASTRID Security Controller v.2

riceve una richiesta sull'endpoint "/register/event", ma non è la sola condizione, in quanto per la terza regola abbiamo una seconda condizione in AND che richiede che il tipo di evento contenuto all'interno dell'interceptionObject sia di tipo "add", mentre per l'ultima regola, l'evento contenuto deve essere di tipo "delete".

Da notare che nelle varie regole vengono prelevati anche diversi componenti, in base all'azione che deve essere svolta.

Grazie all'introduzione delle Drools Rules ora il comportamento del Security Controller in seguito all ricezione di una richiesta su un determinato endpoint, è modificabile senza agire sul codice del modulo stesso, ma semplicemente variando le istruzioni che sono poste all'interno del ramo "then" corrispondente alla condizione di interesse. Il tutto è fattibile senza neanche la ricompilazione del modulo, in quanto le regole vengono lette dal Security Controller a runtime.

Per permettere la modifica di tali regole anche a personale non specializzato, si è cercato di integrare all'interno del progetto l'utilizzo di strumenti che sono già ampiamente noti ed utilizzati da tutti, in quanto, sebbene la modifica delle regole così implementata richieda solamente una semplice correzione ad un file *.drl* che è possibile tramite un qualsiasi programma di elaborazione testuale, anche il più semplice, come Notepad o Blocco note, pensiamo che offrire la possibilità ad un utente non esperto di interagire con il sistema utilizzando un'interfaccia grafica che sia più "guidata" che non un semplice file di testo, possa essere maggiormente di aiuto. Per questo motivo sono state introdotte le tavole di verità, le quali sono offerte da Drools, e che si possono leggere e modificare tramite un file *.xls*, ovvero un file apribile e modificabile utilizzando il noto software Microsoft Excel o altri programmi simili (di cui esistono alternative anche open source come OpenOffice e molte altre).

Nella Figura 5.8 possiamo quindi vedere una parte della tavola della verità introdotta (non possiamo riportare l'intera tavola per motivi di spazio insufficiente). Su tale file (che si chiama quindi *interception.xls*) sono state trasferite le regole Drools del nostro Security Controller. All'avvio del modulo, ora le regole vengono estratte da tale file, la cui modifica risulta molto più semplice rispetto al file di testo precedente.

Nelle prima riga vediamo la definizione delle classi da importare e che verranno usate nelle definizione delle regole. La tabella di verità vera propria inizia dalla riga 6. Nella prima colonna (A) si inseriscono i nomi delle regole, mentre le successive colonne possono essere delle condizioni oppure delle azioni. In corrispondenza di una cella vuota, quella condizione/azione viene ignorata per quella regola. Ecco quindi che, ad esempio, considerando la seconda regole (riga 12, "Register Policy"), se l'interceptionRequest contiene come campo "command" la stringa *"/register/-policy"* (la seconda cella è vuota quindi quella condizione viene ignorata per questa regola), e come *component1* abbiamo "Verefoo", mentre come *component2* abbiamo il ContextBroker, (la cella corrispondente al component3 è vuota quindi viene ignorata), allora le azioni svolte sono quelle contenute nella cella H9, in quanto la cella H12 risulta non vuota (mentre le azioni contenute nella cella G9 non vengono svolte in quanto la cella G12 è vuota).

Se in futuro si volesse aggiungere delle regole, magari per gestire altri tipi di eventi che giungono al Security Controller, è necessario solamente inserire nuove regole a tale tabella di verità.

5.5 Salvataggio eventi all'interno di un database Neo4j

l'ASTRID Security Controller ha il compito di reagire a determinati eventi che si verificano all'interno dell'infrastruttura di rete e che gli vengono notificati, ad esempio all'aggiunta di nuove istanze di servizi, oppure alla loro terminazione. In via prettamente sperimentale, in quanto non richiesto dalle specifiche di sviluppo del Security Controller, si è pensato di archiviare tutti gli eventi gestiti, all'interno di un database a grafo, quale Neo4j (la cui spiegazione è stata fornita nel capitolo 3.5), in modo da mantenere una storicità degli eventi gestiti.

A questo scopo si è creata una classe di servizio (che prende il nome di *Neo4jService*), la quale ha il compito di istanziare un collegamento con il database Neo4j, oltre a gestire le comunicazioni con esso. I parametri per contattare il database Neo4j sono forniti grazie al file di configurazione *Astrid-components.xml*, dove è necessario indicare per il database Neo4j un componente il cui attributo "name" sia pari a *Astrid_DB*.

Il costruttore di tale classe (raffigurato nel Listato 5.15), accetta come parametri l'URL e la porta su cui raggiungere il database Neo4j, oltre al nome utente e alla password da utilizzare per l'accesso. Una volta ricevute queste informazioni, tenta di istanziare un collegamento con il database esterno, se tutto si conclude positivamente, all'interno della variabile *driver* avremo il canale pronto per l'invio delle query all'Astrid_DB; in caso contrario la variabile driver viene posta a null.

```
1 public Neo4jService(String url, String port, String username, String password) {
2     try {
3         driver = GraphDatabase.driver("bolt://" + url + ":" + port, AuthTokens.
  ↪ basic(username, password));
4     }catch(Exception e) {
5         driver = null;
6     }
7 }
8 }
```

Listato 5.15. Costruttore della classe *Neo4jService*

All'interno di questa classe è presente anche il metodo utilizzato per il salvataggio degli eventi che si verificano nell'infrastruttura, all'interno del database Neo4j. Tale metodo è raffigurato nel listato seguente:

```
1 public boolean writing(String EventType, String resourceType, String Name, String
  ↪  ip, String uid) {
2     try {
3         Session session = driver.session();
4         session.writeTransaction(tx -> tx.run("CREATE (:InfrastructureEvent {Type
  ↪ : '" + EventType + "}) - [:Event] -> (:EventData {resourceType: '" +
  ↪ resourceType +
5                 "', name: '" + Name + "', ip: '" + ip + "', uid: '" + uid +"'})")
  ↪ );
6     }catch (Exception e) {
7         return false;
8     }
```

```
9       return true;
10  }
11  }
```

Listato 5.16. Implementazione del metodo *writing*, utilizzato per l'invio delle query d'inserzione al database Neo4j

Il metodo in esame riceve i dettagli dell'evento ricevuto e, tramite il driver, prova a inviare la query d'inserimento al database Neo4j esterno. Tale metodo fornisce un feedback al chiamante di tipo booleano: se tutto si conclude positivamente viene ritornato 'vero', altrimenti si ritorna 'falso', in modo da far conoscere al chiamante l'esito dell'operazione di inserimento e che possa quindi agire di conseguenza.

Tabella 5.1. API REST implementate nella prima versione del Security Controller

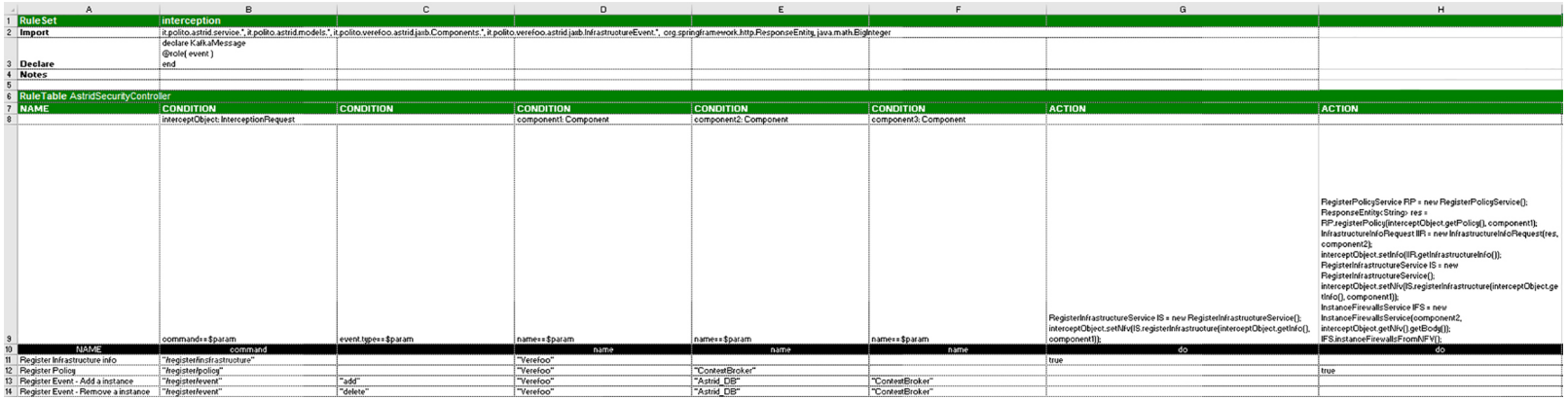
API	**Metodo**	**Corpo**	**Descrizione**
registerInfrastructure	POST	Grafo della rete in formato XML	Tale API riceve le informazioni sull'infrastruttura in formato XML, le invia al modulo esterno VEREFOO, attende i risultati e li fornisce in risposta
registerPolicy	POST	Politiche di raggiungibilità da applicare alla rete, in formato TEXT PLAIN	Tale API riceve le politiche da applicare sul grafo di rete in formato testo semplice e le invia al modulo esterno VEREFOO. Attende la risposta e la inoltra al mittente
registerEvent	POST	Evento accaduto nella rete, in formato XML	Tale API riceve, in formato XML, l'evento che è accaduto nella rete e lo inoltra a VEREFOO

Tabella 5.2. Regole impostate da Verefoo sui firewall con IP 2.208.56.31, 2.208.56.32, 2.208.56.33

IP sor.	**IP dest.**	**Porta sor.**	**Porta dest.**	**Prot.**	**Azione**
2.208.56.*	2.208.56.*	*	*	*	CONSENTI
	*	*	*	*	NEGA

Tabella 5.3. Regole impostate da Verefoo sul firewall con IP 2.208.56.34

IP sor.	**IP dest.**	**Porta sor.**	**Porta dest.**	**Prot.**	**Azione**
2.208.56.77	2.208.56.74	*	*	*	CONSENTI
2.208.56.74	2.208.56.77	*	*	*	CONSENTI
	*	*	*	*	NEGA



	A	B	C	D	E	F	G	H
1	**RuleSet**	**interception**						
2	**Import**	it.polito.astrid.service.*, it.polito.astrid.models.*, it.polito.verefoo.astrid.jaxb.Components.*, it.polito.verefoo.astrid.jaxb.InfrastrutureEvent.*, org.springframework.http.ResponseEntity, java.math.BigInteger						
3	**Declare**	declare KafkaMessage @role(event) end						
4	**Notes**							
5								
6	**RuleTable** AstridSecurityController							
7	**NAME**	**CONDITION**	**CONDITION**	**CONDITION**	**CONDITION**	**CONDITION**	**ACTION**	**ACTION**
8		interceptObject: InterceptionRequest		component1: Component	component2: Component	component3: Component		

9		command== $param	event.type== $param	name== $param	name== $param	name== $param	RegisterInfrastructureService IS = new RegisterInfrastructureService();
							interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo(), component1));
								RegisterPolicyService RP = new RegisterPolicyService();
								ResponseEntity<String> res =
								RP.registerPolicy(interceptObject.getPolicy(), component1);
								InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(res, component2);
								interceptObject.setInfo(IIR.getInfrastructureInfo());
								RegisterInfrastructureService IS = new RegisterInfrastructureService();
								interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo(), component1));
								InstanceFirewallsService IFS = new InstanceFirewallsService(component2, interceptObject.getNfv().getBody());
								IFS.instanceFirewallsFromNFV();

10	NAME	command	name	name	name	name	do	do
11	Register Infrastructure info	"/registerInfrastrutoure"		"Verefoo"			true	
12	Register Policy	"/register/policy"		"Verefoo"	"ContestBroker"			true
13	Register Event - Add a instance	"/registerEvent"	"add"	"Verefoo"	"Astrid_DB"	"ContestBroker"		
14	Register Event - Remove a instance	"/registerEvent"	"delete"	"Verefoo"	"Astrid_DB"	"ContestBroker"		

Figura 5.8. Tavola della verità delle regole Drools

Capitolo 6

Implementazione

In questo capitolo daremo una spiegazione generale del modulo Context Broker e della sua importanza per l'intera architettura ASTRID; a seguire illustreremo le operazioni che sono state effettuate per l'interfacciamento del nostro modulo Security Controller con il modulo Context Broker, spiegando anche la necessità di questa operazione. Nell'ultima sezione di questo capitolo, invece, illustreremo come l'Astrid Security Controller svilluppato è in grado di reagire ad attacchi per cui riceve notifica.

6.1 Introduzione al Context Broker

Come si è visto dalla Figura 4.1, l'ASTRID Context Broker è un componente centrale dell'architettura del sistema, snodo fondamentale da cui transitano e sono immagazzinate le varie informazioni che i diversi moduli devono scambiarsi per poter permettere il funzionamento dell'intero sistema.

Nello specifico, il modulo Context Broker, altro non è che un database, il quale immagazzina tutta una serie di informazioni che vengono utilizzate dai vari moduli. Per poter astrarre il database dall'architettura che lo circonda, e senza la necessità di fornire troppi dettagli tecnici all'esterno del database stesso, si è pensato di creare un modulo che avesse il compito esclusivo di interagire con questo database e che fornisse i dati richiesti a tutti gli altri moduli. Ecco quindi che, quando si cita il Context Broker, invece di riferirsi al vero ASTRID Context Broker che sarebbe il database, ci si riferisce a questo modulo secondario, il quale si interfaccia sia con il database che con i moduli esterni per fornire le informazioni richieste. Tale modulo secondario prende il nome di Astrid Context Broker Manager (ma che in realtà verrà sempre chiamato impropriamente Astrid Context Broker). Nella figura 6.1 possiamo vedere il Data Model dell'ASTRID Context Broker Manager.

Cerchiamo ora di dare una breve spiegazione dell'architettura del Context Broker di modo che possano poi risultare più chiare le operazioni che eseguiremo nel seguito,

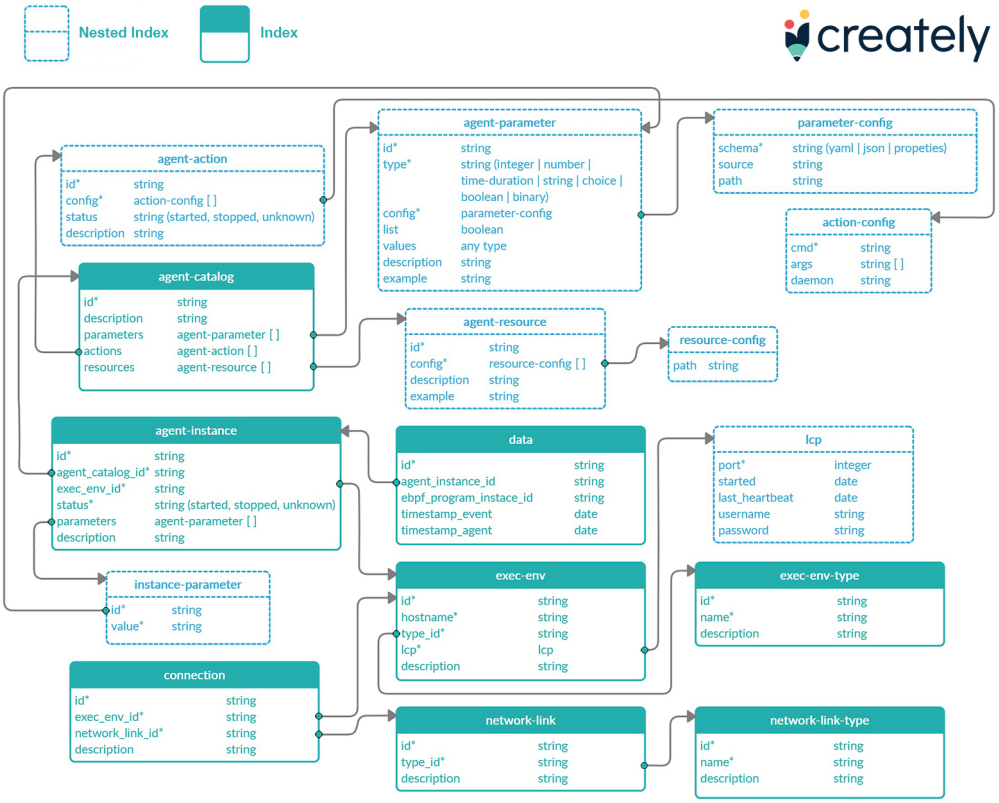


Figura 6.1. Data model dell'ASTRID Context Broker Manager - fonte: [20]

in particolare le varie query che saranno utilizzate per interrogare il database.

Come abbiamo già illustrato più volte, la nostra rete si compone di servizi virtuali (NFV), i quali sono istanziati in POD. Ogni servizio che compone la nostra infrastruttura viene immagazzinato all'interno del Context Broker come *exec-env*, il quale si caratterizza da un id (il quale deve essere univoco), da un hostname, da un type_id, da una descrizione e da un lcp, il quale è un tipo di dato personalizzato che descrive i parametri necessari per l'accesso a quel servizio, come la porta e le credenziali necessarie. E' possibile istanziare innumerevoli tipi di servizi all'interno della rete, ad esempio un database my-sql, un firewall, e molti altri. L'*exec-env-type* rappresenta tutti i tipi di servizi che è possibile utilizzare all'interno della rete; essi vengono caratterizzati da un id, da un nome e da una descrizione.

Le varie infrastrutture di rete sono invece immagazzinate come *network-link*, caratterizzate da un id, da un type_id e da una descrizione. Il campo type_id rappresenta la chiave esterna che determina il tipo di infrastruttura di rete (connessione Point-to-Point, Ethernet, ecc...). Tutti i vari tipi di infrastruttura di rete sono immagazzinati in *network-link-type*, e caratterizzati da un id, da un nome e

da una descrizione.

L'appartenenza dei vari *exec-env* ad una rete è determinata da *connection*: ogni *exec-env* che appartiene ad un determinato *network-link* ha un record all'interno di *connection*, composto da un id, dall'exec_env_id che è l'id dell'exec-env a cui si riferisce, e dal network_link_id, la quale è la chiave esterna che si riferisce al network-link; l'ultimo campo presente è una stringa di descrizione. risulta quindi evidente che tutti gli exec_env_id che hanno il medesimo network_link_id appartengono alla stessa infrastruttura di rete; in altre parole, sono tutti servizi che fanno parte della medesima rete.

Gli ASTRID Agents (raffigurati nella Figura 4.1), ovvero i moduli che sono istanziati all'interno dei vari servizi di rete per monitorarne il funzionamento ed eseguire le azioni che l'ASTRID Security Controller ordina, possono essere di vari tipi: primo fra tutti, possono essere dei servizi Firewall (tipo utilizzato in questa tesi), ma possono anche essere degli IDS (Intrusion Detection System)[1], dei VPN Gateway[2] ma anche molto altro. L'elenco di tutti questi servizi che possono essere istanziati come ASTRID Agents è contenuto in *agent-catalog*, caratterizzati da un id, da un descrizione, da una lista di parametri necessari per quel tipo di servizio, da una lista di azioni da svolgere e di risorse. Sarà cura dell'ASTRID Context Broker instanziare i servizi richiesti, impostandone i parametri settati e anche eseguirne le azioni.

Ogni ASTRID Agents che deve essere istanziato all'interno della rete presenta un record in *agent-instance*, caratterizzato da un id, dall'agent_catalog_id di cui quel servizio ne è un'istanza, dall'exec_env_id che rappresenta il servizio su cui quell'ASTRID Agents è in esecuzione, lo stato (se è attivo, spento o in uno stato sconosciuto), la lista di parametri che caratterizza quell'istanza e una descrizione.

Il modulo ASTRID Context Broker dialoga con gli altri moduli utilizzano API REST, attraverso cui è possibile inviare interrogazioni ed ottenere le informazioni richieste. Illustriamo nella Tabella 6.1 le API REST che sono offerte da tale module, ma solamente quelle necessarie all'ASTRID Security Controller per eseguire le varie operazioni che dovrà svolgere (le API REST offerte dall'ASTRID Context Broker sono assai numerosi, ed è per questo motivo che non possiamo elencarle in modo esaustivo nella presente tesi, innanzitutto perchè non è il nostro obiettivo, inoltre esso è solamente un mezzo attraverso cui il Security Controller raggiunge i suoi obiettivi).

[1]Gli Intrusion Detection System, in reti virtualizzate come il nostro ambito, sono moduli software che analizzano il traffico di rete per identificare possibile traffico anomalo che potrebbe significare un'intrusione di terzi non autorizzati all'interno della rete.

[2]i VPN-Gateway sono dei punti terminali delle connessioni VPN (Virtual Private Network), punto di incontro tra la rete pubblica e il punto di ingresso/uscita della rete privata

Tabella 6.1: API REST offerte dal modulo ASTRID Context Broker Manager, utilizzate dall'ASTRID Security Controller

EXEC-ENV

Endpoint	/exec-env/[id]
Metodo	GET

Corpo

Query in formato JSON (nell'esempio riportato viene selezionato l'exec-env che ha l'id uguale a <exec-env-id>):

```
"select": [ "hostname" ],
    "where": {
        "equals": {
            "target:" "id",
            "expr": "<exec-env-id>"
        }
    }
```

Descrizione

API utilizzata per interrogare il Context Broker ed ottenere la lista di tutti gli exec-env (se il corpo della richiesta è vuoto), oppure solo gli exec-env che soddisfano la query inserita nel corpo della richiesta. E' possibile ottenere il dettaglio di uno specifico exec-env inviando una richiesta con corpo vuoto ed inserendone l'id direttamente nel link dell'endpoint

Endpoint:	/exec-env
Metodo:	POST

Corpo

Query d'inserimento in formato JSON:

```
"id": "<exec-env-id>",
"description": "<description>",
"type_id": "<exec-env-type-id>",
"hostname":"<ip-address>",
"lcp": {
    "port": "<lcp-port>"
}
```

Descrizione

API con la quale è possibile inserire un nuovo exec-env

Endpoint:	/exec-env/[id]
Metodo:	DELETE

Corpo

Query per l'eliminazione di un exec-env in formato JSON (nell'esempio riportato si elimina l'exec-env che ha il campo id = <exec-env-id>):

```
"where": {
    "equals": {
        "target:" "id",
        "expr": "<exec-env-id>"
    }
}
```

Descrizione

API utilizzata per l'eliminazione di uno o più exec-env (nel caso in cui la query restituisca più risultati). Se il corpo della richiesta è vuoto, si eliminano tutti gli exec-env. Per eliminare uno specifico exec-env, è possibile inserire nel link dell'endpoint il suo id

AGENT-INSTANCE

Endpoint: /instance/agent/[id]
Metodo: GET

Corpo

Query in formato JSON (nell'esempio riportato viene selezionato l'agent-instance che ha l'id uguale a <agent-instance-id>):

```
"select": [ "parameters" ],
    "where": {
    "equals": {
        "target:" "id",
        "expr": "<agent-instance-id>"
    }
}
```

Descrizione

API utilizzata per interrogare il Context Broker ed ottenere la lista di tutti gli agent-instance (se il corpo della richiesta è vuoto), oppure solo gli agent-instance che soddisfano la query inserita nel corpo della richiesta. E' possibile ottenere il dettaglio di uno specifico agent-instance inviando una richiesta con corpo vuoto ed inserendone l'id direttamente nel link dell'endpoint

Endpoint: /instance/agent/
Metodo: POST

Corpo

Query d'inserimento in formato JSON:

```
"id": "<agent-instance-id>",
"agent_catalog_id": "<agent-id>",
"exec_env_id": "<exec-env-id>",
"status": "<status>",
"operations": [
    "parameters": [
        {
            "id": "<parameter-id>",
            "value": "<parameter-value>"
        }
    ],
    "actions": [
        {
            "id": "<action-id>",
            "mode": "<action-mode-value>"
        }
    ]
]
```

Descrizione

API con la quale è possibile inserire un nuovo agent-instance

---

Endpoint: /instance/agent/[id]
Metodo: PUT

Corpo

Query d'aggiornamento in formato JSON (nell'esempio si aggiorna l'agent-instance che ha come id = <agent-instance-id>, aggiungendo i parametri e le azioni riportati rispettivamente in "parameters" e "actions"):

```
"id": "<agent-instance-id>",
"operations": [
    "parameters": [
        {
            "id": "<parameter-id>",
            "value": "<new-parameter-value>"
        }
    ],
    "actions": [
        {
            "id": "<action-id>",
            "mode": "<new-action-mode-value>"
        }
    ]
]
```

Descrizione

API con la quale si aggiorna un agent-instance, per modificarne qualche campo, oppure per inserire nuove azioni e/o parametri

---

Endpoint: /instance/agent/[id]
Metodo: DELETE

Corpo

Query per l'eliminazione di un agent-instance in formato JSON (nell'esempio riportato si elimina l'agent-instance che ha il campo id = <agent-instance-id>):

```
"where": {
    "equals": {
        "target:" "id",
        "expr": "<agent-instance-id>"
    }
}
```

Descrizione

API utilizzata per l'eliminazione di uno o più agent-instance (nel caso in cui la query restituisca più risultati). Se il corpo della richiesta è vuoto, si eliminano tutti gli agent-instance. Per eliminare uno specifico agent-instance, è possibile inserire nel link dell'endpoint il suo id

6.2 Progettazione e analisi dei flussi comunicativi

Ora che abbiamo capito l'importanza dell'ASTRID Context Broker, le sue funzioni, quali informazioni può fornire e come ricavarle, possiamo illustrare i flussi comunicativi necessari per raggiungere i nostri obiettivi, oltre a spiegarne anche la loro implementazione.

La Figura 4.1 rappresenta molto bene, in modo schematico, tutti i flussi comunicativi in cui l'ASTRID Security Controller è coinvolto; lo scopo di questa sezione è quello di analizzare nel dettaglio tali flussi.

6.2.1 Ricezione delle regole di raggiungibilità

Come abbiamo già più volte affermato, il gestore della rete interagisce con il sistema attraverso l'ASTRID Security Dashboard (modulo facente parte di ASTRID, ma esterno al Security Controller). Da tale Dashboard è possibile instanziare reti virtuali con i rispettivi servizi NFV, oltre a gestirli. Nella Figura 6.2 è illustrato lo schema comunicativo generale che si scatena quando il gestore della rete richiede all'infrastruttura di calcolare l'allocazione ottima dei firewall e che andiamo ora ad illustrare:

1. Quando viene richiesta la creazione di una nuova rete, la dashboard comunica le informazioni necessarie all'ASTRID Context Broker, creando gli *exec-env* necessari, oltre alla *network-link* e alle *connection*. Se viene richiesto, sempre tramite la dashboard, l'attivazione di politiche di raggiungibilità, ecco che entra in azione l'ASTRID Security Controller;

2. Il gestore della rete, tramite la dashboard, richiede il calcolo ottimale dell'allocazione dei firewall. Per fare questo, la dashboard gli richiede di fornire le policy di raggiungibilità che vuole impostare all'interno della rete definita. Abbiamo già visto, nei capitoli precedenti, esempi di policy di raggiungibilità,

quindi se è il caso, invitiamo il lettore a rivedere il Listato 5.2. Tali policy vengono inviate come corpo in una richiesta API REST all'endpoint *registerPolicy* dell'ASTRID Security Controller;

3. Alla ricezione delle regole di raggiungibilità, l'ASTRID Security Controller richiede le informazioni necessarie all'ASTRID Context Broker per poter costruire l'infrastruttura di rete nel formato richiesto dal modulo Verefoo;

4. Una volta ricevute le informazioni richieste e costruita l'infrastruttura di rete, il Security Controller invia tali informazioni come corpo di una richiesta API REST al modulo esterno Verefoo, a cui precedentemente aveva fornito anche le regole di raggiungibilità;

5. Verefoo calcola la configurazione ottimale dei firewall che garantiscono le regole imposte e, una volta fatto, consegna tali informazioni al Security Controller;

6. A questo punto, l'ASTRID Security Controller deve istanziare tali firewall all'interno della rete e procedere alla loro configurazione. Per fare questo intrattiene una serie di comunicazioni con l'ASTRID Context Broker per istanziare, configurare e inizializzare tali firewall.

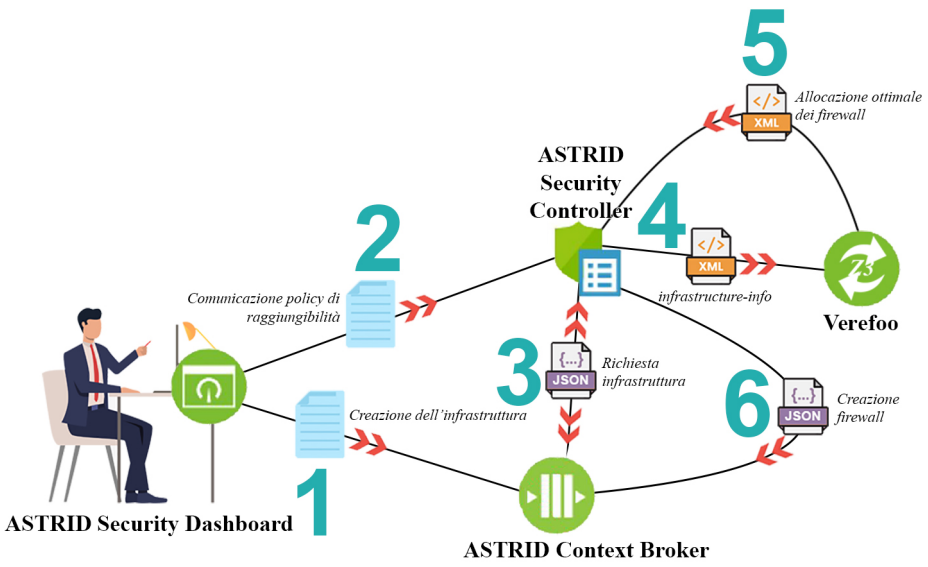


Figura 6.2. Schema comunicativo logico necessario per il calcolo e l'allocazione ottimale dei firewall

Visto lo schema generale delle comunicazioni necessarie, illustriamo ora, con maggior dettaglio, come esse sono state realizzate, incentrandoci ovviamente sull'ASTRID Security Controller e tralasciando le comunicazioni che i moduli esterni

hanno fra di loro, in quanto non ne conosciamo i dettagli non essendo di nostra competenza.

Quando l'ASTRID Security Controller riceve una richiesta sull'enpoint */register/policy*, grazie al framework Spring, tale richiesta viene gestita dal metodo `public ResponseEntity<NFV> registerPolicy(@RequestBody String policy)`, già illustrato nel Listato 5.12. Tale metodo costruisce l'InterceptionRequest inserendo all'interno la stringa dell'endpoint su cui si è ricevuta la richiesta (quindi "/register/policy") e la policy ricevuta (la quale è contenuta come stringa nel corpo della richiesta). Una volta costruita l'InterceptionRequest, essa viene passata al motore delle regole di Drools, insieme ai moduli necessari per la gestione della richiesta, che sono il modulo Verefoo ed il Context Broker (riga 8 del listato 5.12). Il motore delle regole esegue il matching con le regole impostate nel file Excel per trovare la regola corrispondente da eseguire; nel caso in esame la regola che verrà eseguita è quella scritta alla riga 12 della tabella della verità illustrata nella Figura 5.8. Il codice che verrà eseguito (impostato come azione della regola menzionata) è il seguente:

```
1 RegisterPolicyService RP = new RegisterPolicyService();
2 ResponseEntity<String> res = RP.registerPolicy(interceptObject.getPolicy(),
  ↪ component1);
3 InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(res, component2);
4 interceptObject.setInfo(IIR.getInfrastructureInfo());
5 RegisterInfrastructureService IS = new RegisterInfrastructureService();
6 interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo(),
  ↪ component1));
7 InstanceFirewallsService IFS = new InstanceFirewallsService(component2,
  ↪ interceptObject.getNfv().getBody());
8 IFS.instanceFirewallsFromNFV();
```

Listato 6.1. Azione della regola *Register Policy*

La prima azione svolta è la creazione di un'istanza della registerPolicyService, classe in cui avviene l'azione di invio della policy al modulo esterno Verefoo, svolta nel metodo `public ResponseEntity<String> registerPolicy(String policy, Component verefoo)`, che illustriamo qui di seguito:

```
1  public ResponseEntity<String> registerPolicy(String policy, Component verefoo)
   ↪ throws VerefooException, IOException {
2      logger.info("++++++++++ registerPolicy Controller Obtained Policy");
3
4      HttpHeaders headers = new HttpHeaders();
5      headers.setContentType(MediaType.TEXT_PLAIN);
6
7      RestTemplate restTemplate = new RestTemplate();
8
9      // Data attached to the request.
10     HttpEntity<String> requestBody = new HttpEntity<>(policy, headers);
11
12     logger.info("++++++++++ registerPolicy Controller Sending to Verekube");
13     ResponseEntity<String> result = null;
14     try {
15         // Send request with POST method.
16         result = restTemplate.postForEntity("http://" + verefoo.getIPAddress() +
   ↪ ":" + verefoo.getPort() + "/verefoo/graph", requestBody, String.class);
17         // Code = 200.
```

```
18          logger.info("++++++++++ registerPolicy Controller Success from Verekube:
    ↪ " + result.getBody());
19      }catch(HttpServerErrorException | HttpClientErrorException e) {
20          logger.error("++++++++++ error while contacting Verefoo module: " + e.
    ↪ getMessage() + " - " + e.getLocalizedMessage());
21          throw new VerefooException(e.getMessage());
22      }catch (Exception e) {
23          throw new IOException(e.getMessage());
24      }
25  return result;
26  }
```

Listato 6.2. Codice del metodo *registerPolicy*

A tale metodo, visto che il suo compito è l'invio della policy ricevuta al modulo esterno Verefoo, è necessario passare come parametri, oltre alla policy stessa, anche il *component1*, il quale è il modulo Verefoo (lo si può vedere nella tavola della verità, nella cella D12, dove come component1 è selezionato il componente che ha nome uguale a "Verefoo").

Nel corpo del metodo si costruisce l'header della richiesta che verrà poi inviata a Verefoo, oltre ad istanziare un RestTemplate e a notificare le varie azioni svolte sul file di logger. Alla riga 16 viene effettuato l'invio della policy e la risposta viene catturata all'interno della variabile *result*; da notare che l'indirizzo a cui contattare il modulo Verefoo e la porta sono ricavati dal componenete che si riceve come parametro. Se tutto si è svolto correttamente e da Verefoo riceviamo una risposta positiva (200 OK), allora viene segnalato il successo dell'azione nel file di log e si ritorna la risposta che è giunta (riga 18). Nel caso in cui si siano verificati problemi sia lato client che server, allora si catturano le eccezioni che si sono scatenate (che possono essere di due tipi: HttpServerErrorException o HttpClientErrorException), si segnala tale fallimento nel file di log e si ritorna (riga 20). Infine, se dovessero esserci altri problemi durante l'invio (quindi si verifica il lancio di qualche altra eccezione), nel blocco catch di riga 22 si catturano tutte le eccezioni e se ne lancia una del tipo *IOException* inserendo il messaggio di errore generato, la quale viene poi gestita dai metodi di @ExceptionHandler di Spring.

Una volta inviata la policy a Verefoo, l'ASTRID Security Controller deve richiedere le informazioni necessarie al Context Broker per poter costruire *l'infrastructure-info* da inviare a Verefoo per la generazione dell'allocazione ottimale dei firewall. Tale compito viene svolto dalla classe InfrastructureInfoRequest creata appositamente per svolgere tale azione. Il costruttore di tale classe accetta 2 parametri: il primo è la risposta giunta da Verefoo quando si è inviata la policy, e il secondo è il component2, il quale è il Component che ha nome uguale a "ContextBroker" (come si può vedere dalla tavola della verità nella cella E11) . Nella riga 3 del Listato 6.1 si crea quindi un'istanza della classe InfrastructureInfoRequest, mentre nella riga successiva si esegue il metodo `public InfrastructureInfo getInfrastructureInfo ()` qui di seguito riportato:

```
1 public InfrastructureInfo getInfrastructureInfo() throws
  ↪ AstridComponentNotFoundException, IOException, ResourceNotFoundException,
  ↪ JSONException {
2     InfrastructureInfo network = null;
3     if(resultPolicy.getStatusCode() == HttpStatus.OK) {
4         String name = ((resultPolicy.getBody()).substring(resultPolicy.getBody().
  ↪ indexOf(":")+1)).trim();
5         logger.info("+++++++++ ask to Context Broker the infrastructure info of
  ↪ graph: " + name);
6         network = getInfrastructureInfoFromCB(name, ContextBroker);
7     }
8     return network;
9 }
```

Listato 6.3. Codice del metodo *getInfrastructureInfo*

Tale metodo non riceve parametri, ma estrae semplicemente il nome del grafo della rete di cui è necessario richiedere le informazioni sui componenti (in quanto il nome del grafo è contenuto nella policy che è stata inviata precedentemente a Verefoo; quest'ultimo nella risposta ha integrato anch'esso il nome del grafo, quindi ora non ci resta che estrarre tale informazione). Una volta ottenuta l'informazione, si inserisce essa nel file di log, e poi si invoca, alla riga 6, un metodo interno alla classe *InfrastructureInfoRequest* il quale si occupa di creare le query necessarie per la generazione dell'infrastruttura di rete. Vediamo nel dettaglio il corpo di tale metodo, il quale si compone di circa una novantina di righe di codice:

```
1 private InfrastructureInfo getInfrastructureInfoFromCB(String graphName,
  ↪ Component CB) throws AstridComponentNotFoundException, IOException,
  ↪ JSONException, ContextBrokerException {
2     InfrastructureInfo net = new InfrastructureInfo();
3     ObjectMapper objectMapper = new ObjectMapper();
4     int i=0;
5
6     //create request to get the connection into the network
7     HttpHeaders headers = new HttpHeaders();
8     headers.setContentType(MediaType.APPLICATION_JSON);
9
10    RestTemplate restTemplate = new RestTemplate();
11    Jaxb2RootElementHttpMessageConverter converter = new
  ↪ Jaxb2RootElementHttpMessageConverter();
12    converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
13    restTemplate.setMessageConverters(Arrays.asList(converter, new
  ↪ StringHttpMessageConverter()));
14    // Data attached to the request.
15
16    ResponseEntity<String> result = null;
17    try {
18        // Send request with GET method
19        result = restTemplate.getForEntity("http://" + CB.getIPAddress() + ":" +
  ↪ CB.getPort() + "/connection", String.class);
20        logger.info("+++++++++ Context Broker answer successfully with
  ↪ connection of requested network");
21    }catch(HttpServerErrorException e) {
22        logger.error("+++++++++ error while contacting Context Broker module: "
  ↪ + e.getMessage());
23        throw new IOException();
24    }catch (HttpClientErrorException e) {
25        logger.error("+++++++++ Context Broker answer with an error: " + e.
  ↪ getMessage());
```

```
26          throw new ContextBrokerException("Context Broker answer with an error: "
↪ + e.getMessage());
27      }
28      //create a list with connections received
29      List<Connection> connection = objectMapper.readValue(result.getBody(), new
↪ TypeReference<List<Connection>>(){});
30
31      List<Execution_Environment> exec_env = new ArrayList<>();
32      ResponseEntity<String> ee = null;
33      //ask to CB to get the exec_env details with that ID
34      for (i=0;i<connection.size();i++){
35          if(connection.get(i).getNetwork_link_id().compareTo(graphName)==0){
36              try {
37                  ee = restTemplate.getForEntity("http://" + CB.getIPAddress() + ":
↪ " + CB.getPort() + "/exec-env/" + connection.get(i).getExec_env_id(), String.
↪ class);
38                  exec_env.add(objectMapper.readValue(ee.getBody().substring(1, ee.
↪ getBody().length()-1), Execution_Environment.class));
39              } catch(HttpServerErrorException e) {
40                  logger.error("+++++++++ error while contacting Context Broker
↪ module: " + e.getMessage());
41                  throw new IOException();
42              }catch(HttpClientErrorException e) {
43                  logger.error("+++++++++ Context Broker answer with an error
↪ while obtain exec-env: " + e.getMessage());
44                  throw new ContextBrokerException("Context Broker answer with an
↪ error while obtain exec-env: " + e.getMessage());
45              }
46          }
47      }
48
49      //now we can create the infrastructureInfo to send as answer
50      Metadata meta = new Metadata();
51      meta.setName(graphName);
52      net.setMetadata(meta);
53
54      Spec spec = new Spec();
55      Iterator<Service> it = spec.getService().iterator();
56      i=0;
57      Service s;
58      Port p;
59      Instance ins;
60      boolean found = false;
61      for(i=0;i<exec_env.size();i++){
62          found = false;
63          it = spec.getService().iterator();
64          while(it.hasNext() && found == false) {
65              Service ser = it.next();
66              if(ser.getName().equals(exec_env.get(i).getType_id())){
67                  //this is a new instance of an existing service
68                  found = true;
69                  ins = new Instance();
70                  ins.setIp(exec_env.get(i).getHostname());
71                  ins.setUid(exec_env.get(i).getId());
72                  ser.getInstance().add(ins);
73              }}
74              if(found == false) {
75                  //this is a new service
76                  s = new Service();
77                  p = new Port();
78                  ins = new Instance();
79                  s.setName(exec_env.get(i).getType_id());
80                  p.setInternal(exec_env.get(i).getLcp().getPort().shortValue());
```

```
81                  s.setPort(p);
82                  ins.setIp(exec_env.get(i).getHostname());
83                  ins.setUid(exec_env.get(i).getId());
84                  s.getInstance().add(ins);
85                  spec.getService().add(s);
86              }
87      }
88      net.setSpec(spec);
89      return net;
90  }
```

Listato 6.4. Codice del metodo *getInfrastructureInfoFromCB*

Il metodo riceve due parametri: il nome della rete per la quale si vogliono ricevere le informazioni ed il componente Context Broker, verso cui indirizzare le richieste. Il compito del metodo in esame è di costruire ed inviare le query d'interrogazione necessarie per costruire l'*infrastructure-info* da consegnare poi a Verefoo. Nel Listato 5.3 è raffigurato un esempio di *infrastructure-info*. Per costruire tale XML è necessario innanzitutto conoscere tutti gli *exec-env* che fanno parte del grafo d'interesse; per avere questa informazione bisogna inviare una richiesta di tipo GET all'ASTRID Context Broker, indirizzata all'endpoint *connection*, la quale ci fornisce l'elenco di tutti nodi contenuti in *connection* (riga 19). Un esempio di ciò che può essere contenuto nella risposta a tale richiesta, è riportato nel Listato A.3 nell'appendice tecnica.

Tali nodi vengono sistemati in una lista (riga 29), sfruttando l'oggetto *ObjectMapper*. E' importante notare che la richiesta che viene inviata al Context Broker non contiene alcun corpo, in quanto utilizzando il framework Spring, non è possibile inserire nessun corpo alle richieste di tipo GET (come da convenzioni standard). Per questo motivo non è possibile inserire una query che filtri i risultati in modo tale da restituirci solo i dati che ci interessano (nel nostro caso, sono di interesse tutte le connessioni che hanno come campo *network_link_id* l'id del grafo), ma verranno restituiti tutte le connessioni presenti; sarà poi compito nostro, all'interno del metodo, filtrare i risultati ottenuti.

Per costruire l'infrastructure-info è necessario ricevere tutte le informazioni sugli *exec-env* che compongono il grafo d'interesse; per questo motivo creiamo una seconda lista (riga 29) il cui scopo sarà quello di contenere tali informazioni utili.

Nella riga 34 eseguiamo quindi un ciclo, all'interno del quale si vanno ad inviare tante richiesta GET all'endpoint *exec-env/[id]* del Context Broker, quante sono le connessioni che sono presenti nel grafo d'interesse (questo filtro viene eseguito con l'istruzione *if* posta alla riga 35). Alla ricezione di ogni risposta, sempre grazie all'oggetto *ObjectMapper*, viene creata un'istanza della classe Execution_Environment la quale modella l'exec-env, e la si inserisce all'interno della lista *exec_env*, creata precedentemente.

Ora abbiamo tutte le informazioni necessarie per la costruzione dell'*infrastructure-info*, la quale inizia a riga 50, dove si crea l'oggetto *Metadata*, il quale contiene il

nome del grafo. Nelle righe successive si scandisce tutta la lista *exec-env* per andare a creare ogni nodo dell'infrastruttura di rete, andando a controllare se, per ogni nodo, quel tipo di servizio è già presente nella rete; in caso affermativo significa che quel nodo è un'ulteriore istanza di quel servizio (caso verificato dall'istruzione *if* posta a riga 66), altrimenti esso rappresenta un nuovo servizio da inserire nell'infrastruttura (verificato sempre dall'*if* posto alla riga 74).

Una volta creati i vari servizi e le istanze, la creazione dell'infrastruttura di rete è terminata; essa viene quindi ritornata al metodo illustrato nel Listato 6.3, il quale semplicemente la ritorna al corpo della regola riportato nel Listato 6.1, e che viene quindi inserita all'interno dell'oggetto *interceptionObject*.

E' importante notare che, anche nel metodo appena illustrato, si è cercato di riservare particolare attenzione ai problemi che potrebbero nascere durante il suo svolgimento, ovvero alle eccezioni che è possibile che si scatenino, cercando di gestirle in modo da garantire comunque la prosecuzione dell'esecuzione, benchè l'azione venga interrotta. Un altro aspetto che si è cercato di mantenere sempre in considerazione è il file di log: per ogni azione che viene svolta con successo, ma soprattutto per ogni azione che invece si conslude in modo negativo, si è cercato di fornire i dettagli del problema all'interno del file di log, in modo tale che il personale tecnico addetto al controllo e manutenzione del sistema possa essere aiutato da esso.

Alla riga 5 nel corpo dell'azione della regola (Listato 6.1) viene istanziata la classe *RegisterInfrastructureService*, la quale contiene il metodo necessario all'invio dell'*infrastructure-info* al modulo esterno Verefoo (azione che viene svolta alla riga 6). Il metodo che esegue tale azione è il seguente: `public ResponseEntity<NFV> registerInfrastructure( InfrastructureInfo info, Component verefoo)`. Tale metodo riceve come parametri l'*InfrastructureInfo* calcolata precedentemente, e il componente Verefoo, a cui inviare tale informazione. Nel listato seguente possiamo quindi vedere il codice del metodo:

```
1  public ResponseEntity<NFV> registerInfrastructure(InfrastructureInfo info,
   ↪ Component verefoo) throws ResourceNotFoundException, IOException {
2      if (info == null || info.getMetadata() == null || info.getMetadata().getName
   ↪ () == null) {
3          logger.info("++++++++++ registerInfrastructure Empty body");
4          throw new ResourceNotFoundException("registerInfrastructure Empty body");
5      }
6
7      HttpHeaders headers = new HttpHeaders();
8      headers.setContentType(MediaType.APPLICATION_XML);
9      logger.info("++++++++++ registerInfrastructure Controller Obtained
   ↪ Infrastructure info for: "
10             + info.getMetadata().getName());
11     RestTemplate restTemplate = new RestTemplate();
12     Jaxb2RootElementHttpMessageConverter converter = new
   ↪ Jaxb2RootElementHttpMessageConverter();
13     converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
14     restTemplate.setMessageConverters(Arrays.asList(converter, new
   ↪ StringHttpMessageConverter()));
15     // Data attached to the request.
16     HttpEntity<InfrastructureInfo> requestBody = new HttpEntity<
   ↪ InfrastructureInfo>(info, headers);
```

```
17      logger.info("++++++++++ registerInfrastructure Controller Sending
    ↪ Infrastructure info to Verekube");
18      // Send request with POST method.
19      ResponseEntity<NFV> result=null;
20      try {
21           result = restTemplate.postForEntity("http://" + verefoo.getIPAddress() +
    ↪ ":" + verefoo.getPort() + "/verefoo/dc", requestBody, NFV.class);
22      } catch (HttpServerErrorException | HttpClientErrorException ex) {
23          logger.info("++++++++++ registerInfrastructure Controller  Policy is not
    ↪ defined for this graph");
24          throw new ResourceNotFoundException("Policy is not defined for this graph
    ↪ : "+ex.getMessage() );
25      } catch (Exception e) {
26          logger.info("++++++++++ error while contatcting Verefoo module: " + e.
    ↪ getMessage());
27          throw new IOException();
28      }
29
30      return result;
31  }
```

Listato 6.5. Codice del metodo *registerInfrastructure*

Nelle righe iniziali vediamo che viene eseguito un controllo per accertarsi che le informazioni ricevute siano corrette, dopodichè viene creato l'header, il *RestTemplate* e il convertitore per l'invio delle informazioni al modulo esterno. L'invio delle informazioni a Verefoo avviene alla riga 21, le quali vengono inviate ell'endpoint *verefoo/dc*. Il metodo si conclude con la restituzione di quanto ricevuto come risposta da Verefoo, il quale è un oggetto del tipo personalizzato NFV. Un esempio di tale risposta è stata rappresentata nel Listato A.1.

La risposta fornita da Verefoo viene inserita all'interno dell'*interceptionObject*, ed infine viene svolta l'ultima azione, ovvero quella di istanziare i firewall necessari all'interno del Context Broker. A tale scopo, nella riga 7 del Listato 6.1 viene istanziata la classe *ùService*, la quale contiene all'interno il metodo necessario per la lettura dell'NFV ricevuto da Verefoo e per la successiva istanziazione dei firewall necessari. Il metodo che esegue tali azioni è il seguente: `public void instanceFirewallsFromNFV()`, il quale non riceve alcun parametro in quanto nel costruttore della classe è già stato fornito il *component2* (che è il modulo Context Broker) e la risposta avuta da Verefoo (ovvero l'NFV). Riportiamo qui di seguito il codice del metodo in esame:

```
1  public void instanceFirewallsFromNFV() throws ContextBrokerException{
2      //recover the node that are firewalls from NFV
3      Iterator<Graph> it = graph.getGraphs().getGraph().iterator();
4      int i = 0;
5      Date dateNow = new Date();
6      SimpleDateFormat ft = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss");
7      List<Agent_Instance> agentInstance = new ArrayList<>();
8      RestTemplate restTemplate = new RestTemplate();
9      Jaxb2RootElementHttpMessageConverter converter = new
    ↪ Jaxb2RootElementHttpMessageConverter();
10     converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
11     restTemplate.setMessageConverters(Arrays.asList(converter, new
    ↪ StringHttpMessageConverter()));
12     HttpHeaders headers = new HttpHeaders();
13     headers.setContentType(MediaType.APPLICATION_JSON);
14     HttpEntity<String> requestBody = null;
```

```
15      ResponseEntity<String> result = null;
16      JSONObject query = null;
17      JSONObject defAction = null;
18      List<JSONObject> rules = new ArrayList<>();
19      JSONObject rule = null;
20      JSONArray actions = null;
21      Graph schema = it.next();
22      Iterator<Node> it_node = schema.getNode().iterator();
23      Iterator<Elements> it_ele = null;
24      logger.info("+++++++++ started to instance the firewalls");
25
26
27      //recover the agent instance
28      agentInstance = getAgentInstance();
29      while(it_node.hasNext()) {
30          Node n = it_node.next();
31          update=false;
32          if(n.getFunctionalType().toString().compareTo("FIREWALL") == 0) {
33              //check if a firewall already exist of this node
34              logger.info("+++++++++ check if firewall of node with ID: " + n.
    ↪ getConfiguration().getName() + " already exist");
35              if (checkFirewallExist(n.getConfiguration().getName(), agentInstance)
    ↪ ) {
36                  //firewall already exist, so update it
37                  update=true;
38                  logger.info("+++++++++ firewall of node with ID: " + n.
    ↪ getConfiguration().getName() + " already exist. Updating in progress...");
39                  try {
40                      query.put("id", "firewall@" + n.getConfiguration().getName())
    ↪ ;
41                  }catch(Exception ex) {
42                      logger.error("+++++++++ error while costruct query: " + ex.
    ↪ getMessage());
43                      throw new ContextBrokerException(ex.getMessage());
44                  }
45              }else {
46                  //there isn't a firewall configured
47                  logger.info("+++++++++ firewall of node with ID: " + n.
    ↪ getConfiguration().getName() + " not exist. Creating in progress...");
48                  try {
49                      query.put("id", "firewall@" + n.getConfiguration().getName())
    ↪ ;
50                      query.put("agent_catalog_id", "firewall");
51                      query.put("exec_env_id", n.getConfiguration().getName());
52                      query.put("status", "stopped");
53                  }catch(Exception ex) {
54                      logger.error("+++++++++ error while costruct query: " + ex.
    ↪ getMessage());
55                      throw new ContextBrokerException(ex.getMessage());
56                  }
57              }
58
59              //create actions
60              //check and set the default action
61              if(n.getConfiguration().getFirewall().getDefaultAction() != null){
62                  defAction = new JSONObject();
63                  actions = new JSONArray();
64                  try {
65                      defAction.put("id", "default");
66                      defAction.put("action", n.getConfiguration().getFirewall().
    ↪ getDefaultAction());
67                      defAction.put("timestamp", ft.format(dateNow));
68                      actions.put(defAction);
```

```
69                  }catch(Exception ex) {
70                      logger.error("+++++++++ error while costruct the query: " +
↪ ex.getMessage());
71                      throw new ContextBrokerException(ex.getMessage());
72                  }
73              }
74              //insert into the query the rules
75              it_ele = n.getConfiguration().getFirewall().getElements().iterator();
76              i = 0;
77              rules.clear();
78              //create the rules
79              while(it_ele.hasNext()){
80                  Elements ele = it_ele.next();
81                  rule = new JSONObject();
82                  try {
83                      rule.put("id", "insert");
84                      rule.put("n", "rule" + (i+1) + "@" + n.getConfiguration().
↪ getName());
85                      rule.put("src", getAddressWithMask(ele.getSource()));
86                      rule.put("dst", getAddressWithMask(ele.getDestination()));
87                      rule.put("action", ele.getAction());
88                      rule.put("timestamp", ft.format(dateNow));
89                      actions.put(rule);
90                      i++;
91                  }catch(Exception ex) {
92                      logger.error("+++++++++ error while costruct the query: " +
↪ ex.getMessage());
93                      throw new ContextBrokerException(ex.getMessage());
94                  }
95              }
96
97              try {
98                  query.put("actions", actions);
99              }catch(Exception ex) {
100                 logger.error("+++++++++ error while costruct the query: " + ex.
↪ getMessage());
101                 throw new ContextBrokerException(ex.getMessage());
102             }
103             //send the request
104             requestBody = new HttpEntity<String>(query.toString(), headers);
105             try {
106                 if(update==false) {
107                     result = restTemplate.postForEntity("http://" + ContextBroker
↪ .getIPAddress() + ":" + ContextBroker.getPort() + "/instance/agent",
↪ requestBody, String.class);
108                     logger.info("+++++++++ Firewall successfully instanced!");
109                 }else {
110                     restTemplate.put("http://" + ContextBroker.getIPAddress() + "
↪ :" + ContextBroker.getPort() + "/instance/agent", requestBody);
111                     logger.info("+++++++++ Firewall successfully updated!");
112                 }
113             }catch (HttpClientErrorException e) {
114                 if(e.getStatusCode() == HttpStatus.UNPROCESSABLE_ENTITY) {
115                     logger.info("+++++++++ Unable to reach the firewall, but
↪ insert it into Context Broker");
116                 }else {
117                     logger.error("+++++++++ error while instance the firewalls: "
↪  + e.getMessage());
118                     throw new ContextBrokerException(e.getMessage());
119                 }
120             }catch (HttpServerErrorException e) {
121                 logger.error("+++++++++ error while instance the firewalls: " + e
↪ .getMessage());
```

```
122                 throw new ContextBrokerException(e.getMessage());
123             }
124         }
125     }
126 }
```

Nelle prime 24 righe si dichiarano le variabiali che saranno poi utilizzate nel corso delle operazioni. E' importante ricordare che i firewall nell'architettura ASTRID non sono altro che ASTRID Agent Instance; la prima operazione da svolgere, quindi, è il recupero degli Agent Instance già attivi nella rete. Tale operazione viene svolta da un altro metodo: `private List<Agent_Instance> getAgentInstance()`, richiamato alla linea 28, ed il cui codice è qui di seguito riportato:

```
1  private List<Agent_Instance> getAgentInstance() throws ContextBrokerException {
2      List<Agent_Instance> agentInstance = new ArrayList<>();
3      ObjectMapper objectMapper = new ObjectMapper();
4      RestTemplate restTemplate = new RestTemplate();
5      Jaxb2RootElementHttpMessageConverter converter = new
   ↪ Jaxb2RootElementHttpMessageConverter();
6      converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
7      restTemplate.setMessageConverters(Arrays.asList(converter, new
   ↪ StringHttpMessageConverter()));
8      ResponseEntity<String> result = null;
9      try {
10         result = restTemplate.getForEntity("http://" + ContextBroker.getIPAddress
   ↪ () + ":" + ContextBroker.getPort() + "/instance/agent", String.class);
11         agentInstance = objectMapper.readValue(result.getBody(), new
   ↪ TypeReference<List<Agent_Instance>>(){});
12     }catch(HttpServerErrorException ex) {
13         logger.error("+++++++++ error while recover the Agent instance from
   ↪ Context Broker: " + ex.getMessage());
14         throw new ContextBrokerException(ex.getStatusCode().toString());
15     } catch (HttpClientErrorException e) {
16         if(e.getStatusCode() != HttpStatus.NOT_FOUND) {
17             logger.error("+++++++++ error while recover the Agent instance from
   ↪ Context Broker: " + e.getMessage());
18             throw new ContextBrokerException(e.getMessage());
19         }
20     } catch (JsonParseException e) {
21         logger.error("+++++++++ error while recover the Agent instance from
   ↪ Context Broker: " + e.getMessage());
22         throw new ContextBrokerException(e.getMessage());
23     } catch (JsonMappingException e) {
24         logger.error("+++++++++ error while recover the Agent instance from
   ↪ Context Broker: " + e.getMessage());
25         throw new ContextBrokerException(e.getMessage());
26     } catch (IOException e) {
27         logger.error("+++++++++ error while recover the Agent instance from
   ↪ Context Broker: " + e.getMessage());
28         throw new ContextBrokerException(e.getMessage());
29     }
30     return agentInstance;
31 }
```

La prima operazione svolta è la creazione di una lista con lo scopo di ospitare gli *Agent Instance*, subito dopo ci sono le operazioni comuni che sono già state più volte illustrate, al fine di creare gli strumenti necessari per l'invio delle interrogazioni al

Context Broker (*RestTemplate* e *convertitore*) e per eseguire l'unmarshalling[3] delle informazioni ricevute come risposta, utilizzando l'oggetto *ObjectMapper*. Alla riga 10 viene effettuata la richiesta all'endpoint */instance/agent* del Context Broker con il metodo GET, dalla quale si riceve come risposta tutta li lista dei ASTRID Agent Instance già istanziati. Ricevuta la risposta, alla riga 11, viene effettuato l'unmarshalling per inserire nella lista tutti gli elementi ricevuti, prima di concludere il metodo ritornando tale lista.

Una volta che il metodo `public void instanceFirewallsFromNFV()` ha ricevuto la lista degli ASTRID Agent Instance, alla riga 29 inizia il ciclo per andare ad analizzare ogni nodo presente nell'NFV ricevuto da Verefoo. Per i nostri scopi, ci interessano solamente i nodi il cui campo *functional_type* è settato a "FIREWALL" (controllo che viene eseguito alla riga 32). La prima azione svolta sui nodi che soddisfano tale requiesito è la verifica se quel firewall è già presente come Agent Instance nel grafo in oggetto; tale verifica viene svolta con il supporto di un metodo creato per tale scopo (metodo la cui firma è `private boolean checkFirewallExist(String exec_env_id, List<Agent_Instance> ↪ ai)` il quale riceve come parametri l'*exec_env_id* del nodo che vogliamo testare, e la lista degli Agent Instance già presenti), di cui però non riportiamo il codice in quanto molto semplice (si tratta di una semplice iterazione sui vari elementi della lista e di un blocco *if* per testarne l'uguaglianza). Nel caso in cui tale Agent Instance sia già presente, è necessario andare ad aggiornare le sue regole (in quanto magari hanno subito delle variazioni). A questo scopo si utilizza un flag (update), il quale viene settato a "vero" se si tratta di un aggiornamento, in quanto esso sarà utile successivamente per la costruzione della query (la quale sarà diversa se si tratta di un inserimento di un firewall, oppure di un aggiornamento delle regole); se invece il controllo di presenza ha esito negativo, allora si continua alla riga 48 con la creazione della query d'inserimento dell'Agent Instance: Nel primo blocco *try* si assegnano i seguenti campi:

- Riga 49 - Si assegna al campo *id* la stringa composta da "firewall@" + il *configuration name* del nodo in esame;
- Riga 50 - Si assegna al campo *agent_catalog_id* l'id dell'Agent a cui ci si riferisce, il quale è ovviamente settato a *firewall*;
- Riga 51 - Si assegna al campo *exec_env_id* l'id dell'exec_env a cui il firewall si riferisce, ricavandolo dal *configuration name* del nodo in esame;
- Riga 52 - Come *status* del firewall si pone "stopped" per segnalare che tale Agent Instance non è ancora in esecuzione;

[3]con il termine *unmarshalling* ci si riferisce all'operazione di creazione di un oggetto Java a partire dalla sua rappresentazione XML

Per inserire le regole all'interno del firewall istanziato è necessario conoscere quale azione permette di eseguire tale operazione. Tutte le azioni che un certo ASTRID Agent supporta, sono riportate all'interno dell'*ASTRID Agent Catalog*. Per quanto riguarda il firewall, tutte le azioni ad oggi supportate sono riportate nel Listato A.4 all'interno dell'appendice tecnica.

Per imporre le regole che ci sono state comunicate da Verefoo, noi faremo uso dell'azione "default" e dell'azione "insert". Dal listato precedente, vediamo che per ogni azione supportata è necessario fornire anche dei parametri che caratterizzano l'azione stessa. Tali parametri sono indicati all'interno delle parentese graffe.

Alla riga 65 vediamo l'impostazione del comportamento di default, cioè come si deve comportare il firewall in mancanza di specifica regola. E' ora il momento di inserire nella query tutte le regole di raggiungibilità che il firewall deve applicare; per fare questo si estraggono gli "elements" dal nodo in esame e, per ognuno di essi, si inserisce una regola all'interno della query, la quale viene composto come segue:

- Riga 83 - Si assegna al campo *id* l'id dell'azione che deve essere svolta per inserire la regola all'interno del firewall; esso sarà quindi impostato pari a "insert";

- Riga 84 - Assegnamo al campo *n* la stringa "rule" seguita da un numero che altro non è che un contatore numerico. Viene poi inserito il simbolo "@" seguito dal *configuration name*;

- Riga 85 e 86 - Vengono riportati rispettivamente gli indirizzi IP sorgente e destinazione pacchetti per cui si deve applicare tale regola. L'indirizzo lo si estrae dal nodo in esame con il metodo *getSource* per il mittente, oppure *getDestination* per la destinazione, ma siccome gli indirizzi forniti da Verefoo contengono un "-1" ad indicare la maschera, esso deve essere convertito nel metodo d'indicazione standard (ovvero con l'indirizzo Ip seguito da "/8" o "/16", ecc). A tal fine è stato creato un metodo che esegue questa conversione, la cui firma è `private String getAddressWithMask(String ip)` (non riportiamo l'implementazione del metodo in quanto è molto semplice da riprodurre);

- Riga 87 - Si inserisce l'azione che deve compiere il firewall per tali flussi, ovvero consentirli (quindi il campo sarà pari ad ALLOW) oppure negarli (quindi il campo sarà pari a DENY). Questo campo sarà pari al campo "Action" dell'elements in esame;

- Riga 88 - Nel campo *timestamp* si inserirà la data e l'ora in cui viene creata la regole, quindi data e ora correnti.

La query d'inserimento è ora completa e pronta per essere inviata al Context Broker. Nel listato A.5 vediamo un esempio di una query d'inserimento.

Alla riga 107 viene quindi effettuato l'invio della richiesta all'endpoint */instance/agent* con il metodo POST, inserendo nel corpo della richiesta la query d'inserimento. Per quanto riguarda la gestione delle eccezioni, in questo invio si è ritenuto non un errore la risposta 422 - UNPROCESSABLE_ENTITY, in quanto se il Context Broker risponde con tale codice di stato significa che non è stato in grado di istanziare il firewall, ma che comunque l'inserimento è avvenuto con successo.

E' evidente, quindi, che avremo l'invio di tante richieste quante sono le istanze di firewall da inserire.

Con questo metodo si è conclusa l'analisi del primo flusso realizzato, che possiamo quindi trovare riassunto nell'immagine 6.3.

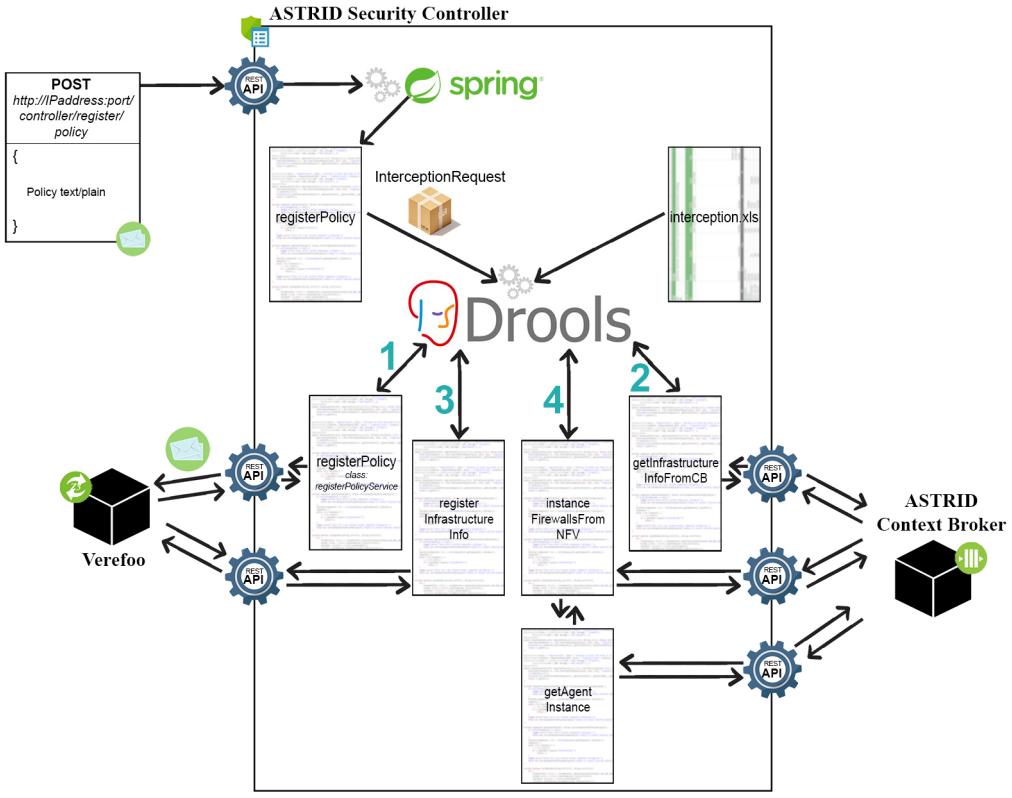


Figura 6.3. Schema riassuntivo del flusso implementativo che si scatena a seguito della ricezione di policy di raggiungibilità

6.2.2 Ricezione di eventi di creazione di nuove istanze nell'infrastruttura

Nel capitolo 4.1 abbiamo illustrato i compiti in carico all'ASTRID Security Controller. Il suo primo compito, ovvero quello di riceve le politiche di raggiungibilità e fornire la configurazione ottimale dei firewall per garantire tali politiche è svolto dal flusso che abbiamo illustrato poc'anzi.

Un secondo compito che deve svolgere l'ASTRID Security Controller è quello di reagire automaticamente ad eventi che gli vengono notificati, comportandosi secondo regole prestabilite a priori ma che possono subire delle variazioni. Ci occuperemo in questa sezione di ciò che è necessario fare quando il Security Controller riceve una notifica di aggiunta di un'istanza di un servizio, in una determinata rete: essendo che siamo in ambienti virtualizzati, quando la domanda di un determinato servizio è crescente, l'orchestratore dell'infrastruttura crea automaticamente una nuova istanza del servizio congestionato, in modo da garantire il bilanciamento del traffico. Il flusso che verrà analizzato nella prossima sezione riguarderà la rimozione di un'istanza; essendo un flusso molto simile a quello che andremo ad illustrate ora, lo schema qui presente sarà valido anche per la rimozione di un'istanza. All'atto della creazione di questa nuova istanza, è necessario inviare una notifica all'ASTRID Security Controller, in quanto anche questa nuova istanza dovrà rigorosamente rispettare le politiche di raggiungibilità imposte dal gestore della rete per quel tipo di servizio. Illustiamo nella Figura 6.4 lo schema comunicativo di tale flusso.

1. All'atto della creazione o rimozione di un'istanza di un servizio all'interno dell'infrastruttura, l'orchestratore invia tale aggiornamento all'ASTRID COntext Broker;

2. Conseguentemente, sempre l'orchestratore, invia l'evento di aggiornamento all'ASTRID Security Controller, tramite l'endpoint *register/event* con una richiesta di tipo POST ed inserendo nel corpo di tale richiesta l'evento accaduto in formato XML (un esempio di tale evento è raffigurato nel Listato 6.6);

3. l'ASTRID Security Controller invia le richieste necessarie alla costruzione dell'*infrastructure-info* all'ASTRID Context Broker;

4. Ricevute le risposte dal Context Broker, il Security Controller costruisce l'*infrastructure-info* e la invia, in formato XML all'interno del corpo di una richiesta REST API all'endpoint *verefoo/dc*, al modulo esterno Verefoo;

5. Verefoo calcola la configurazione ottimale dei firewall e risponde alla richiesta ricevuta;

6. il Security Controller deve quindi aggiornare la configurazione dei firewall all'interno dell'infrastruttura, seguendo le indicazioni fornite da Verekube, attraverso query in formato JSON.

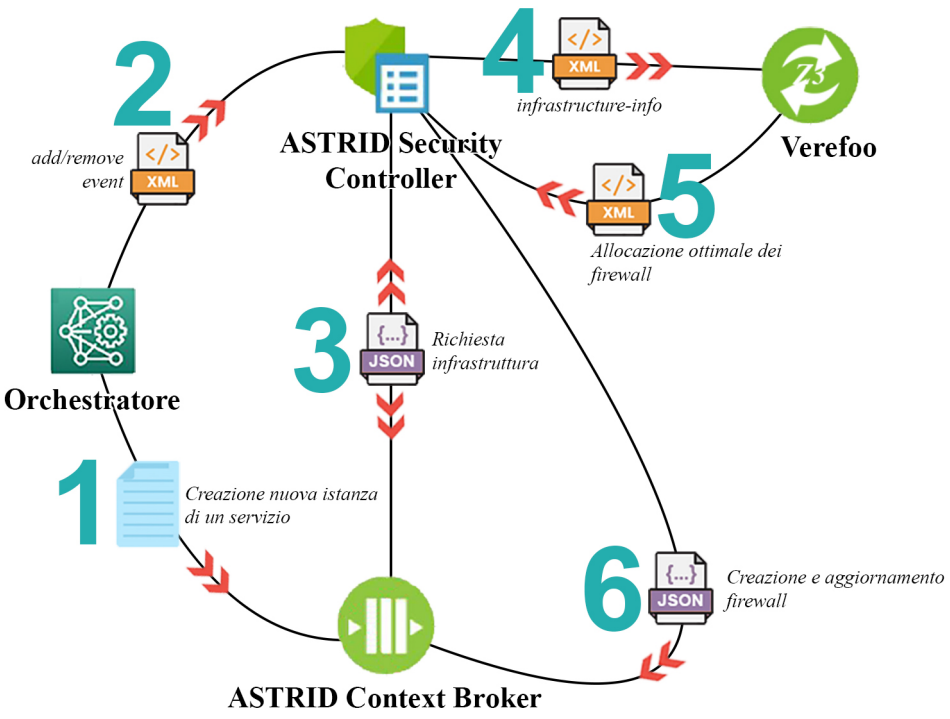


Figura 6.4. Schema comunicativo logico necessario alla gestione di un evento di aggiunta o rimozione istanza accaduto nell'infrastruttura

```
<?xml version="1.0" encoding="UTF-8"?>
<InfrastructureEvent type="add">
    <EventData resourceType="pod" name="apache" ip="172.20.1.241" uid="apache-
↪ c7d97955-2dtf6"></EventData>
    <EventTime>2019-06-28T13:55:01.546188824Z</EventTime>
</InfrastructureEvent>
```

Listato 6.6. Esempio di XML inviato all'endpoint *register/event* dell'ASTRID Security Controller

Illustriamo ora con maggior dettaglio le operazioni che accadono all'interno dell'ASTRID Security Controller alla ricezione di un evento di aggiunta di qualche istanza di un determinato servizio.

L'orchestratore invia l'informazione sull'evento accaduto tramite una richiesta POST sull'endpoint *register/event* del Security Controller, inserendo nel corpo il dettaglio dell'evento accaduto in formato XML (un esempio è il Listato 6.6). L'ASTRID Security Controller, alla ricezione di una richiesta sull'endpoint specificato, grazie al framework Spring, invoca il metodo `public ResponseEntity<NFV> registerEvent (@RequestBody InfrastructureEvent event)`, illustrato nel Listato 5.6. Il metodo costruisce l'InterceptionRequest inserendo nel campo *command* l'endpoint su cui si è ricevuta la richiesta (quindi "/register/event"), oltre al corpo della richiesta stessa. Infine,

l'oggetto costruito viene passato al motore delle regole Drools, insieme ai componenti necessari per la gestione della richiesta che sono: il modulo Verefoo, il Context Broker e l'Astrid_DB. La regola che verrà eseguita è quella scritta a riga 13 del file Excel illustrato alla Figura 5.8. L'azione di tale regola è la seguente:

```
1 RegisterEventService REES = new RegisterEventService();
2 String graphName = REES.registerEvent(interceptObject.getEvent(), component2);
3 InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(component3);
4 interceptObject.setInfo(IIR.getInfrastructureInfo(graphName));
5 RegisterInfrastructureService IS = new RegisterInfrastructureService();
6 interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo(),
  ↪ component1));
7 InstanceFirewallsService IFS = new InstanceFirewallsService(component3,
  ↪ interceptObject.getNfv().getBody());
8 IFS.instanceFirewallsFromNFV();
```

Listato 6.7. Azione della regola *Register Event - Add a instance*

Osservando le azioni che vengono compiute si nota che, a parte le 2 righe iniziali, le seguenti sono identiche alle azioni svolte dal corpo della regola *Register Policy*, illustrata nel Listato 6.1. Per tale motivo, nel seguito della trattazione di questa sezione, illustriamo il dettaglio dei metodi svolti nelle prime 2 righe del Listato 6.7, la restante parte è già stata trattata ampiamente nella sezione precedente.

Nella prima riga del Listato 6.7 si crea un'istanza della classe *RegisterEventService*, all'interno della quale si esegue il metodo `public String registerEvent(InfrastructureEvent event, Component astrid_db)`, il quale riceve come parametri il dettaglio dell'evento accaduto ed il componente Astrid_DB per effettuare il salvataggio dello stesso. Il codice di tale metodo è qui di seguito illustrato:

```
1  public String registerEvent(InfrastructureEvent event, Component astrid_db){
2      logger.info("++++++++++ registerEvent Controller Obtained Event of type: " +
   ↪ event.getType());
3      if(astrid_db.getIPAddress().compareTo("0.0.0.0")==0) {
4          logger.info("++++++++++ There isn't any Astrid DB configured!");
5      }else {
6          neo4j = new Neo4jService(astrid_db.getIPAddress(), astrid_db.getPort().
   ↪ toString(), astrid_db.getUsername(), astrid_db.getPassword());
7          boolean n = neo4j.writing(event.getType(), event.getEventData().
   ↪ getResourceType(), event.getEventData().getName(), event.getEventData().getIp()
   ↪ , event.getEventData().getUid());
8          if (n==true)
9              logger.info("++++++++++ Event correctly insert into Astrid DB");
10         else
11             logger.error("++++++++++ Error while insert the event into Astrid DB"
   ↪ );
12     }
13     return event.getGraphName();
14 }
```

Listato 6.8. Implementazione del metodo *registerEvent*

Nel codice del metodo rappresentato nel Listato 6.8 si esegue il salvataggio dell'evento accaduto, all'interno del database esterno Astrid_DB, il quale è un database Neo4j (argomento affrontato nel capitolo 3.5). Come già detto, l'inserimento di un database per la memorizzazione dello storico degli eventi accaduti, NON è richiesto dalle specifiche fornite per l'implementazione dell'ASTRID Security Controller;

per tali motivi, quindi, è necessario che il componente funzioni correttamente sia che l'Astrid_DB sia presente, ma anche in mancanza di esso. Ecco quindi che a riga 3 si esegue una verifica della presenza dell'Astrid_DB, controllando l'indirizzo IP del componente 'astrid_db' fornito: se l'indirizzo IP è pari a '0.0.0.0' significa che il database esterno non è configurato (tale indirizzo è settato dal Security Controller stesso all'avvio, nel momento in cui esegue l'unmarshalling del file *Astrid-components.xml*, illustrato nel capitolo 5.2, se non trova alcun componente 'Astrid_DB' settato), quindi il Security Controller notifica tale avvenimento all'interno del file di log e poi termina l'esecuzione di questo metodo, ritornando il nome del grafo; se invece l'IP è diverso, si istanza la classe *Neo4jService* (illustrata nel capitolo 3.5) e si richiede l'inserimento di tale evento all'interno del database Astrid_DB. Fatto questo il metodo termina ritornado sempre il nome del grafo della rete su cui è accaduto tale evento.

Il nome del grafo viene salvato all'interno di una variabile locale (come si vede a riga 2 del corpo della regola riportato nel Listato 6.7), la quale viene poi utilizzata alla riga 4 per richiedere le informazioni necessarie all'ASTRID Security Controller per creare l'*infrastructure-info*, come illustrato nella sezione precedente.

6.2.3 Ricezione di eventi di rimozione istanze nell'infrastruttura

In questa sezione illustreremo il comportamento del Security Controller alla ricezione di un evento di rimozione di un'istanza di un servizio dall'infrastruttura. Come avviene per l'aggiunta di istanze, essendo in ambienti virtualizzati, quando il traffico indirizzato a uno specifico servizio ritorna alla normalità, l'orchestratore termina le istanze aggiuntive. A seguito di ciò, è necessario informare anche il Security Controller in modo da aggiornare la configurazione dei firewall. Lo schema comunicativo è, anche in questo caso, raffigurato nella Figura 6.4.

Anche in questo caso, come nel caso precedente, si riceve l'evento accaduto (che qui sarà di tipo "delete"), sull'endpoint *register/event* del Security Controller, con inserito nel corpo della richiesta il dettaglio dell'evento accaduto in formato XML (un esempio è il Listato 6.6).

Spring invocherà quindi il metodo `public ResponseEntity<NFV> registerEvent(@RequestBody InfrastructureEvent event)`, illustrato nel Listato 5.6 il quale creerà l'InterceptionRequest come descritto nella sezione precedente, e poi passerà l'oggetto al motore delle regole Drools. Sino a qui, non ci sono novità, in quanto ciò che è accaduto è del tutto identico a quanto accaduto per l'evento di aggiunta istanza; la regola che verrà eseguita, però, in questo caso sarà "Register Event - Remove a instance", ovvero la regola scritta a riga 14 della tavola della verità illustrata nella Figura 5.8.

L'azione che viene svolta è rappresentata nel listato seguente:

```
1 RegisterEventService REES = new RegisterEventService();
2 String graphName = REES.registerEvent(interceptObject.getEvent(), component2);
```

```
3  InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(component3);
4  interceptObject.setInfo(IIR.getInfrastructureInfo(graphName));
5  RegisterInfrastructureService IS = new RegisterInfrastructureService();
6  interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo(),
   ↪ component1));
7  RemoveFirewallsService RFS = new RemoveFirewallsService(component3);
8  RFS.removeFirewallfromInfrastructureEvent(interceptObject.getEvent());
9  InstanceFirewallsService IFS = new InstanceFirewallsService(component3,
   ↪ interceptObject.getNfv().getBody());
10 IFS.instanceFirewallsFromNFV();
```

Listato 6.9. Azione della regola *Register Event - Remove a instance*

Tale regola svolge le medesime azioni della regola *Register Event - Add a instance*, oltre ad un'azione in più rappresentata a riga 7 e 8 del listato precedente, che andremo quindi ora ad analizzare. L'azione aggiuntiva che deve essere svolta nel caso in cui si riceve un evento di rimozione di un'istanza è la terminazione e rimozione del firewall istanziato per l'istanza del servizio che è stata cessata. Ecco quindi la necessità di avere queste due righe di codice in più come azione di tale regola. L'azione di rimozione del firewall viene svolta all'interno di una classe creata appositamente per tale scopo: *RemoveFirewallsService*. Il costruttore di tale classe riceve come unico parametro il componente Context Broker. Alla riga 8 dell'azione della regola vediamo che viene eseguito il metodo `public void removeFirewallfromInfrastructureEvent(InfrastructureEvent event)`, il cui codice è qui di seguito rappresentato. Lo scopo di tale metodo è quello di eliminare il firewall legato all'istanza del servizio per cui si è ricevuto l'evento di eliminazione. A seguito di questo, poi, come si vede dalle righe successive dell'azione della regola, si istanzieranno i nuovi firewall aggiornati.

```
1  public void removeFirewallfromInfrastructureEvent(InfrastructureEvent event)
   ↪ throws ContextBrokerException {
2      RestTemplate restTemplate = new RestTemplate();
3      Jaxb2RootElementHttpMessageConverter converter = new
   ↪ Jaxb2RootElementHttpMessageConverter();
4      converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
5      restTemplate.setMessageConverters(Arrays.asList(converter, new
   ↪ StringHttpMessageConverter()));
6      try {
7          restTemplate.delete("http://" + ContextBroker.getIPAddress() + ":" +
   ↪ ContextBroker.getPort() + "/instance/agent/firewall@" + event.getEventData().
   ↪ getUid());
8          logger.info("+++++++++ Firewall with ID: firewall@" + event.getEventData
   ↪ ().getUid() + " successfully deleted!");
9      }catch(Exception e) {
10         logger.error("+++++++++ error while deleting the firewall with id:
   ↪ firewall@" +  event.getEventData().getUid() + " - " + e.getMessage());
11         throw new ContextBrokerException(e.getMessage());
12     }
13 }
```

Listato 6.10. Codice del metodo *removeFirewallfromInfrastructureEvent*

Il metodo in esame riceve come parametro l'evento di rimozione dell'istanza del servizio, su cui deve andare a rimuovere il firewall. Nelle righe iniziali si istanziano gli oggetti necessari per l'invio della richiesta all'Astrid Context Broker, mentre alla riga 7 si invia la richiesta di rimozione del firewall, indirizzata all'endpoint

instance/agent/[id_firewall] dell'Astrid Context Broker. La richiesta inviata è di tipo DELETE.

Le operazioni successive che sono eseguite nel corpo dell'azione della regola *Register Event - Remove a instance*, eseguono la creazione delle nuove istanze dei firewall come descritto nella prima sezione di questo capitolo.

6.3 Notifica di attacco DoS

Una delle più diffuse minacce informatiche è l'attacco DoS - Denial of Service, ovvero interruzione di servizio. Tale attacco è molto semplice da mettere in atto, ed è in grado di mettere fuori uso un'intera infrastruttura di rete, che può essere di un'azienda ma anche di un servizio pubblico fondamentale, come un ospedale.

Questo tipo di attacco consiste nell'inondare di moltissime richieste HTTP fasulle un bersaglio (che quasi sempre sono server), in modo che esso non riesca più a generare risposte per tutti. In questo modo, i veri utenti che necessiterebbero di accedere a quel servizio non sono più in grado di raggiungerlo, in quanto saturo. L'attacco che viene utilizzato più spesso oggi, però, è una variante del DoS, e prende il nome di DDoS. Tale attacco, prevede di colpire il bersaglio da moltissimi punti diversi, ovvero inviando pacchetti non da una singola fonte (come viene fatto in un attacco di tipo DoS), ma inviando contemporaneamente pacchetti da molte fonti diverse al medesimo bersaglio. Utilizzando questa tecnica, si riesce a raggiungere il blocco del servizio molto prima rispetto ad un attacco DoS.

E' però necessario fare una considerazione: l'attacco DDoS prevede che molteplici mittenti contattino contemporaneamente il medesimo bersaglio; questo potrebbe far pensare che comunque l'organizzazione di tale attacco non sia così semplice, ma è esattamente il contrario. Per effettuare attacchi di questo tipo, che vengono sempre eseguiti da criminali informatici, si utilizzano molto spesso le macchine di persone comuni ignari di quello che sta succedendo al proprio dispositivo. Ciò è possibile perchè i cracker hanno a disposizione una vasta rete di computer, infettati precedentemente da apposito software malevole, che risponde ai loro comandi, questo senza che il proprietario si accorga di nulla. Ecco quindi che far partire un attacco di questo tipo diventa davvero una cosa semplice.

La rete di computer che sono infettati da malware e per questo sono sotto controllo dei cracker prende il nome di botnet. La Figura 6.5 mostra uno schema riepilogativo dell'attacco DDoS.

L'attuale versione del Security Controller, sviluppata durante questa tesi, è potenzialmente in grado di reagire anche ad attacchi di tipo DDoS, ma sono stati eseguite test solamente per attacchi di tipo DoS (Capitolo 7.6); pertanto, allo stato attuale, possiamo dichiarare che il Security Controller è per ora in grado di reagire con successo ad attacchi DoS.

Nell'architettura di Astrid, il Security Controller è in ascolto su un topic Kafka, sul quale verranno inviate tutte le notifiche di attacchi, quando essi vengono rilevati.

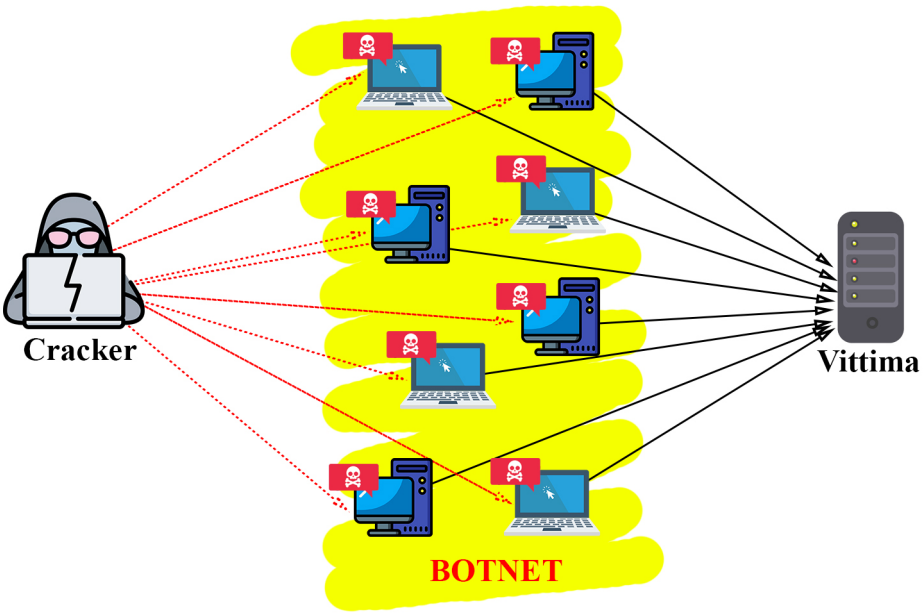


Figura 6.5. Schema logico di un attacco DDoS

In particolare, nella versione fin qui sviluppata, il Security Controller è in ascolto sul topic Kafka 'testing-result' e per farlo si è utilizzato ancora una volta il supporto che ci viene fornito dal framework Spring, in particolare l'annotazione `@KafkaListener`, la quale permette di indicare a Spring quale metodo eseguire alla ricezione di un messaggio su un determinato topic. Nel nostro caso, il metodo implementato per tale scopo è quello riportato qui di seguito:

```
1 @KafkaListener(topics = "testing-result", autoStartup = "${listen.auto.kafka}")
2 public void listen(ConsumerRecord<?, ?> cr) throws Exception {
3       //receive a message on kafka bus
4       ObjectMapper objectMapper = new ObjectMapper();
5       KafkaMessage mess = objectMapper.readValue(cr.value().toString().toLowerCase
  ↪ (), KafkaMessage.class);
6       InterceptionRequest IR = new InterceptionRequest(null, null, null, "kafka",
  ↪ null, null);
7       IR.setMess(mess);
8       droolsService.sendInterceptionRequest(IR, getContextBroker(), null, null);
9  }
```

Listato 6.11. Codice del metodo di gestione di un messaggio Kafka

Nella prima riga si utilizza l'annotazione citata precedentemente, indicando anche per quale topics il metodo deve essere richiamato, oltre ad un parametro che abilita o meno l'ascolto da parte del Security Controller. Quando si riceve un messaggio nel topics 'testing-result', viene quindi chiamato questo metodo, il quale istanzia un ObjectMapper in modo tale da mappare il messaggio ricevuto in un oggetto Java del tipo KafkaMessage. Quest'ultima è una classe creata su modello

del messaggio che si riceve dal topic Kafka, il quale sarà della forma indicata nel Listato seguente:

```
{
    "SOURCE_IP": "172.20.1.5",
    "SOURCE_PORT": 55105,
    "DESTINATION_IP": "172.20.1.150",
    "DESTINATION_PORT": 80,
    "PROTOCOL": "tcp",
    "TIMESTAMP": 1605618710.3542113,
    "ATTACK": "DDoS LOIC"
}
```

Listato 6.12. Struttura dei messaggi che il Security Controller si aspetta di riceve sul Kafka bus

Una volta ottenuto l'oggetto java contenente il messaggio, si crea un oggetto di tipo InterceptionRequest (riga 6), indicando come endpoint la stringa "kafka", e settando al proprio interno il messaggio ricevuto (riga 7). Nell'ultima riga, infine, si inserisce tale InterceptionRequest all'interno del motore delle regole Drools affinchè possa essere gestito. Giunti a questo punto, sono le regole Drools che determinano l'azione che deve essere svolta. Per tale motivo, si sono quindi inserite 2 nuove regole all'interno della tavola della verità:

- Message Attack DoS - count: tale regole viene eseguita quando è presente un oggetto di tipo KafkaMessage all'interno del motore delle regole di Drools ed inoltre, tale messaggio, deve avere il campo ATTACK pari a "DoS LOIC". L'azione che viene eseguita è la semplice notifica di ricezione di un messaggio d'attacco, e l'incremento di un contatore;

- Attack DoS LOIC - countermeasures: questa regola viene eseguita quando il contatore incrementato dalla regole precedentemente citata supera un certo valore soglia che è impostabile come parametri all'interno della tavola della verità stessa (tale valore, di default è impostato su 4). L'azione di questa regola sono le contromisure per la cessazione dell'attacco in corso, e che riportiamo nel Listato seguente.

```
InstanceFirewallsService IFS = new InstanceFirewallsService(component1, null);
IFS.instanceFirewallFromKafkaEvent(k);
```

Listato 6.13. Azione della regola Attack DoS LOIC

Da come si può vedere dalle azioni svolte, si istanzia la classe InstanceFirewallsService, passando come unico parametro il componente Context Broker; successivamente si esegue il metodo *instanceFirewallFromKafkaEvent*, passando come parametro il messaggio Kafka ricevuto.

Per arginare un attacco di questo tipo è necessario avere un firewall che blocchi i numerosi pacchetti in arrivo e/o in partenza, in quanto dobbiamo anche considerare il caso in cui sia uno dei nodi della nostra rete ad inviare i pacchetti verso una vittima esterna.

Nel Listato A.20 riportiamo il codice in cui viene eseguita la contromisura illustrata. Nelle prime 26 righe si istanziano tutte le variabili utilizzate poi nelle operazioni seguenti. E' interessante notare che, a riga 23 e a riga 25, si richiede rispettivamente la lista di tutti gli Agent Instance presenti e degli Execution Environment. Queste operazioni sono necessarie in quanto dobbiamo sapere innanzitutto a quale Execution Environment appartiene l'IP riportato nel messaggio ricevuto, ma anche se è già in esecuzione un firewall su tale Execution Environment (e quindi dobbiamo solamente aggiornare le sue regole) oppure no (ed in questo caso dobbiamo proprio istanziarlo). Alla riga 29 inizia quindi un ciclo che scandaglia tutti gli Execution Environment fino a che non troviamo quello di interesse. L'Execution Environment di interesse è colui che ha IP pari al campo mittente o destinatario riportato nel messaggio Kafka ricevuto. Quando l'abbiamo individuato controlliamo, a riga 37, se esiste già un firewall istanziato su tale Execution Environment; in tal caso recuperiamo le regole già imposte e le inseriamo nella query di aggiornamento che andremo ad inviare al Context Broker. In tale query, infine, andiamo ad indicare una regola aggiuntiva che va' a bloccare il traffico malevole, ovvero se l'Execution Environment in questione è mittente del traffico malevole, la regola che genereremo bloccherà tutto il traffico generato da tale Execution Environment e destinato alla vittima dell'attacco, se invece è la vittima dell'attacco, la regola generata bloccherà tutto il traffico che si riceve dal mittente che ha IP riportato nel messaggio Kafka ricevuto. Queste azioni descritte sono contenuto nelle righe a partire da 39 fino a 112. Dalla riga 114 in poi, invece, sono contenute le azioni eseguite quando non c'è alcun firewall istanziato sull'Execution Environment. In tal caso, l'azione configurata ora prevede di istanziarne uno con la sola regola di default che blocca tutto il traffico.

Capitolo 7

Testing

Una fase finale ma sempre molto importante dello sviluppo di qualsiasi progetto software è la fase di testing di quanto realizzato, per verificare se esso risponde alle esigenze dichiarate e se permette di raggiungere gli obiettivi prefissati. In questo capitolo, quindi, andremo ad effettuare dei test sull'ASTRID Security Controller sviluppato, per verificare se il comportamento tenuto è o meno quello corretto. Nella sezione finale abbiamo anche effettuato un'analisi delle performance del Security Controller al crescere delle regole imposte.

L'ambiente su cui eseguiamo l'ASTRID Security Controller è una macchina Linux, con CPU Intel Xeon E312xx (Sandy Bridge, IBRS update) e con 10GB di memoria RAM.

La configurazione riportata nel file Astrid-components.xml, del Security Controller è la seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <component name='Verefoo' IPAddress='localhost' Port='8085'/>
    <component name='ContextBroker' IPAddress='130.251.17.128' Port='5000'/>
</components>
```

Listato 7.1. contenuto del file Astrid-components.xml durante l'esecuzione dei test

L'infrastruttura di rete che abbiamo istanziato per le nostre prove è rappresentata nell'immagine 7.1. A tale rete abbiamo dato il nome di *test-network*

Per controllare che tale rete sia effettivamente stata istanziata, controlliamo innanzitutto che gli *exec-env* siano contenuti all'interno dell'ASTRID Context Broker; ci aspettiamo che ci siano, eventualmente fra altri: l'*exec-env* del server Apache, l'*exec-env* del server Nginx e l'*exec-env* del server MySql. Per effettuare questo controllo, inviamo una richiesta di tipo GET all'endpoint */exec-env* dell'ASTRID Security Controller (nel nostro caso, quindi, l'indirizzo a cui si invia la richiesta è il seguente: *http://130.251.17.128:5000/exec-env*), utilizzando il software Postman.

Dalla risposta fornita (riportata nel Listato A.6 nell'appendice tecnica) vediamo che tutti e 3 i servizi sono stati correttamente istanziati; accertiamoci ora che

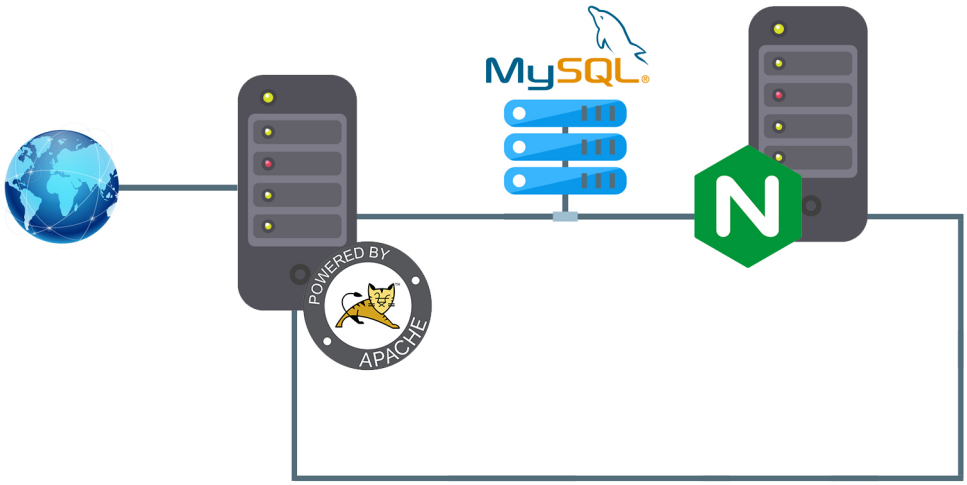


Figura 7.1. Schema della rete virtuale istanziata per eseguire i test

sia stato creato il grafo della nostra rete all'interno di *network-link*, inviando una richiesta sempre di tipo GET all'URL: *http://130.251.17.128:5000/network-link*. Dalla risposta che viene fornita (Listato A.7) abbiamo la conferma che anche il grafo è stato correttamente creato.

Infine, l'ultima verifica che dobbiamo ancora fare è controllare che tutte le istanze che fanno parte della nostra rete, siano state connesse al grafo *test-network*, ovvero che siano presenti all'interno di *connection* un record per ogni *exec-env* che fa' parte della nostra rete. Inviamo quindi una richiesta, di tipo GET, all'URL *http://130.251.17.128:5000/connection*, e riportiamo nel Listato A.8 dell'appendice tecnica la risposta, da cui abbiamo, anche in questo caso, conferma che l'operazione è stata svolta con successo.

Possiamo quindi iniziare i nostri test.

7.1 Test ricezione regole di raggiungibilità

in questa sezione testiamo il comportamento dell'ASTRID Security Controller quando riceve le politiche di raggiungibilità di un grafo (flusso descritto nel dettaglio nel Capitolo 6.2.1).

Per eseguire tale test, quindi, avviamo l'ASTRID Security Controller tramite console e utilizzando i comandi Maven. L'interfaccia che ci si presenta quando il Security Controller è pronto ed in attesa di ricevere dati, è rappresentata nell'immagine 7.2.

Nell'esempio che stiamo svolgendo, vogliamo imporre le seguenti regole di raggiungibilità:

```
kind: Graph
metadata:
    name: test-network
spec:
    policies:
    - from: apache
      to: mysql
      action: forward
    - from: mysql
      to: apache
      action: forward
    - from: nginx
      to: mysql
      action: forward
    - from: mysql
      to: nginx
      action: forward
```

Listato 7.2. Elenco delle policy che viene fornito all'ASTRID Security Controller

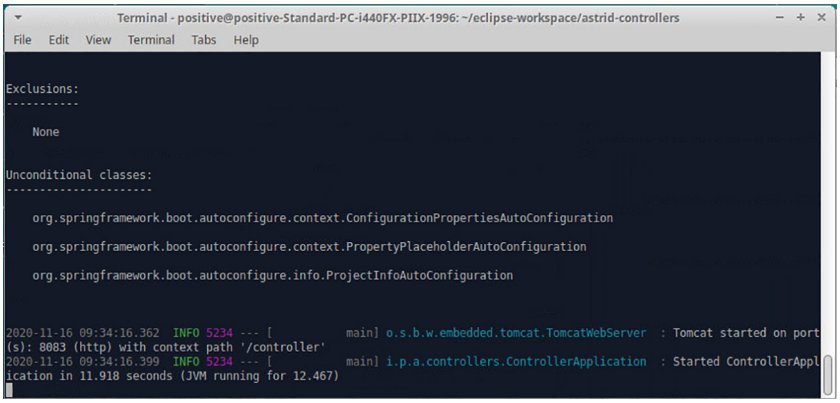


Figura 7.2. Screenshot di ASTRID Security Controller in esecuzione e in attesa di dati

Dalle regole imposte, capiamo che vogliamo rendere possibile la comunicazione dei server apache e nginx con il database mysql e viceversa, ma non vogliamo che i server comunichino fra loro. Per inviare tale policy all'ASTRID Security Controller utilizziamo il software Postman e creiamo una richiesta di tipo POST indirizzata all'URL *http://localhost:8083/controller/register/policy*, inserendo nel corpo, la policy. La risposta che ci viene fornita alla nostra richiesta è riportata nel Listato A.9 dell'appendice tecnica.

Questa risposta è la configurazione ottimale dei firewall, fornita da Verefoo, che garantisce le policy da noi imposte. Analizzando la risposta, vediamo che sono richiesti 3 firewall per garantire le regole di raggiungibilità, ognuno destinato a regolare l'invio/ricezione di dati da uno specifico servizio:

- Per quanto rigarda il server apache (il cui indirizzo IP è 172.20.1.147), è necessario un firewall con indirizzi IP 172.20.1.167, con le regole riportate nella Tabella 7.1 configurate;
- Per il database mysql (il cui indirizzo IP è 172.20.1.150), è necessario un firewall con indirizzi IP 172.20.1.170, con le regole riportate nella Tabella 7.2 configurate;
- Infine, per il server nginx (il cui indirizzo IP è 172.20.1.152), è necessario un firewall con indirizzi IP 172.20.1.172, con le regole riportate nella Tabella 7.3 configurate;

Tabella 7.1. Regole calcolate da Verefoo per il firewall con IP 172.20.1.167 (servizio apache)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.150	172.20.1.147	*	*	*	CONSENTI
172.20.1.147	172.20.1.150	*	*	*	CONSENTI
	*	*	*	*	NEGA

Tabella 7.2. Regole calcolate da Verefoo per il firewall con IP 172.20.1.170 (servizio mysql)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.*	172.20.1.*	*	*	*	CONSENTI
	*	*	*	*	NEGA

Verifichiamo quindi se, l'ASTRID Security Controller, ha istanziato correttamente i firewall necessari all'interno della rete. Per fare questo, utilizziamo sempre il software Postman, e richiediamo l'elenco di tutti gli *agent-instance* presenti, inviano una richiesta di tipo GET all'URL *http://130.251.17.128:5000/instance/agent*. La risposta fornita è riportata nel Listato A.10 dell'appendice tecnica.

Dalla risposta fornita possiamo capire che i firewall sono stati correttamente instanziati e anche le regole per ciascuno di essi sono corrispondenti alle Tabelle 7.1, 7.2 e 7.3. Possiamo ora dare uno sguardo al file di log del Security Controller, giusto per verificare che in esso siano notificati correttamente gli eventi accaduti. Nel listato A.11 riportiamo il contenuto di tale file, e che ora andremo ad analizzare.

Nelle prime righe vediamo che il Security Controller è in ascolto sulla porta 8083. Alla riga 8 vediamo che il Security Controller riceve una richiesta di tipo POST sull'endpoint */controller/register/policy*, la quale viene mappata correttamente nel metodo `public ResponseEntity<NFV> registerPolicy(@RequestBody String policy)`. Il corpo di

Tabella 7.3. Regole calcolate da Verefoo per il firewall con IP 172.20.1.172 (servizio nginx)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.150	172.20.1.152	*	*	*	CONSENTI
172.20.1.152	172.20.1.150	*	*	*	CONSENTI
	*	*	*	*	NEGA

tale richiesta contiene quanto è rappresentato da riga 11 (ovvero le regole di raggiungibilità). A riga 17 abbiamo la notifica di ricezione di tali regole, mentre nella riga successiva abbiamo la conferma che tali regole sono state inviate al modulo esterno Verefoo. Alla riga 19 vediamo l'URL a cui è stata inviata la policy, il cui corpo è raffigurata dalla riga 22.

Nella riga 37 abbiamo ricevuto risposta da Verefoo, la quale viene elaborata dal Security Controller, il quale a riga 39 conferma che Verefoo ha correttamente ricevuto e trattato la policy inviata. L'operazione seguente da effettuare è la richiesta all'ASTRID Context Broker di tutti i servizi che sono istanziati nel grafo in oggetto (tale azione parte alle riga 40 del Listato A.12), che viene effettuata mediante un'interrogazione all'URL *http://130.251.17.128:5000/connection* di tipo GET. A riga 43 abbiamo la notifica di ricev uta della risposta, la quale viene elaborata dal Security Controller, che ci fornisce la correttezza delle operazioni a riga 45. A questo punto, quindi, è necessario recuperare tutte le informazioni sui vari servizi che compongono l'infrastruttura di nostro interesse, per questo partono una serie di interrogazioni indirizzate all'URL *http://130.251.17.128:5000/exec-env/[id-exec-env]* del Context Broker, che vengono notificate a riga 46, 50 e 54. Alla riga 58 abbiamo la notifica da parte dell'ASTRID Security Controller che tutte le richieste sono andate a buon fine e che quindi ora, dopo aver costruito *l'infrastructure-info*, lo si invia a Verefoo (operazione notificata a riga 59). Alla riga 63 abbiamo conferma della ricezione dell'allocazione ottimale dei firewall.

Giunti a questo punto, non resta che istanziare i firewall richiesti da Verefoo. Ecco quindi che nella prima riga del Listato A.13 (riga 65), inizia tale operazione. Alla riga successiva l'ASTRID Security Controller richiede l'elenco di tutti gli ASTRID Agents già presenti, mediante l'invio di una richiesta GET all'URL *http://130.251.17.128:5000/instance/agent*. Una volta ricevuta risposta, iniziano le operazioni per istanziare i firewall richiesti: prima della creazione di ciascun firewall, si controlla se essi siano già presente nell'infrastruttura (operazioni notificate a riga 70, 78 e 86), non essendo presenti nessuno di essi (notifiche a riga 71, 79 e 87), si procede alla loro creazione mediante l'invio di una richiesta POST all'URL *http://130.251.17.128:5000/instance/agent* inserendo nel corpo di tale richiesta la query necessaria per la creazione (riportate nel file di log a riga 74, 82 e 90). Per ciascuna creazione si notifica l'esito (righe 75, 83 e 91), mentre al termine di tutte le

creazioni, si notifica la corretta istanziazione di tutti i firewall (riga 93) e si invia al mittente della richiesta (nel nostro caso al software Postman, ma nell'architettura ASTRID sarà la Dashboard), quanto ricevuto dal modulo esterno Verefoo.

7.2 Test ricezione evento di aggiunta nuova istanza nell'infrastruttura

In questa sezione il nostro obbiettivo è testare il comportamento dell'ASTRID Security Controller alla ricezione di un evento di aggiunta di un'istanza di un servizio già presente e avviato all'interno dell'infrastruttura di rete.

L'evento che si notificherà al Security Controller sarà l'avvio di una seconda istanza del server apache; tale evento rispecchia ciò che potrebbe accadere nella relatà, in quanto, a causa di un picco di traffico le richieste indirizzate al server apache potrebbero diventare molto abbondanti, e questo potrebbe causare un rallentamento delle operazioni o, peggio ancora, sovraccaricare troppo la singola istanza del server e causarne uno shutdown improvviso. L'orchestratore, quindi, crea ed avvia questa nuova istanza nella rete, dopodichè invia l'evento di tipo *add* al Security Controller. Tale evento viene inviato grazie ad una richiesta di tipo POST indirizzata all'URL *http://localhost:8083/controller/register/event*, inserendo nel corpo della richiesta il dettaglio dell'evento, che riportiamo qui di seguito:

```
<InfrastructureEvent type="add" graphName="test-network">
    <EventData resourceType="pod" name="apache" ip="172.20.1.155" uid="apache-58
↪ fc12sdd4-1b6v5"/>
    <EventTime>2020-11-17T08:50:01.546188824Z</EventTime>
</InfrastructureEvent>
```

Listato 7.3. Dettaglio evento di aggiunta istanza inviato all'ASTRID Security Controller

Nel nostro caso, per eseguire il test, non abbiamo a disposizione l'orchestratore della rete, quindi dobbiamo effettuare le operazioni che sarebbero svolte da esso, mediante l'utilizzo del software Postman. Per prima cosa, quindi, creiamo questo nuovo *exec-env*, mediante l'invio della seguente query, come corpo di una richiesta POST indirizzata all'URL *http://130.251.17.128:5000/exec-env*:

```
1  {
2      "id": "apache-58fc12sdd4-1b6v5",
3      "description": "apache server instance",
4      "type_id": "apache",
5      "hostname":"172.20.1.155",
6      "enabled": "true",
7      "lcp": {
8          "port": "80"
9      }
10 }
```

Listato 7.4. Query inviata all'ASTRID Context Broker per l'inserimento dell'exec-env che rappresenta la nuova istanza mysql

Ora effettuiamo il collegamento di questa nuova istanza alla nostra infrastruttura di rete. Per farlo è necessario inviare una richiesta di tipo POST all'URL *http://130.251.17.128:5000/connection*, con il seguente corpo:

```
{
    "description": "apache-58fc12sdd4-1b6v5 network connection",
    "exec_env_id": "apache-58fc12sdd4-1b6v5",
    "id": "apache-58fc12sdd4-1b6v5_connection",
    "network_link_id": "test-network"
}
```

Listato 7.5. Query inviata all'ASTRID Context Broker per l'inserimento della nuova istanza mysql all'interno dell'infrastruttura di rete

Svolte queste operazioni preliminari, possiamo inviare l'evento di notifica (rappresentato nel Listato 7.3) all'ASTRID Security Controller. Dopo circa un minuto, giunge la risposta che riportiamo integralmente nel Listato A.14 dell'appendice tecnica.

Come è già stato illustrato nella sezione precedente, tale risposta rappresenta la configurazione ottimale dei firewall fornita da Verefoo.Facendo un confronto tra questa risposta e la precedente (illustrata nel Listato A.9) si nota che abbiamo l'aggiunta di un firewall, il quale ha il compito di consentire o negare le comunicazioni da e per la nuova istanza del server Apache. Le nuove regole da configurare su ciascun firewall sono riportate nelle tabelle 7.4, 7.5, 7.6, 7.7.

Tabella 7.4. Regole calcolate da Verefoo per il firewall con IP 172.20.1.167 (servizio apache)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.150	172.20.1.147	*	*	*	CONSENTI
172.20.1.147	172.20.1.150	*	*	*	CONSENTI
	*	*	*	*	NEGA

Tabella 7.5. Regole calcolate da Verefoo per il firewall con IP 172.20.1.175 (servizio apache)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.155	172.20.1.150	*	*	*	CONSENTI
172.20.1.150	172.20.1.155	*	*	*	CONSENTI
	*	*	*	*	NEGA

Accertiamoci ora che tutti i firewall siano stati correttamente istanziati, richiedendo l'elenco di tutti gli *agent-instance*. La risposta è riportata nel Listato A.15 dell'appendice tecnica. Da tale risposta abbiamo la conferma che tutti i firewall sono state correttamente istanziati ed anche le regole imposte sono corrette.

Tabella 7.6. Regole calcolate da Verefoo per i firewall con IP 172.20.1.170 (servizio mysql)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.*	172.20.1.*	*	*	*	CONSENTI
	*	*	*	*	NEGA

Tabella 7.7. Regole calcolate da Verefoo per il firewall con IP 172.20.1.172 (servizio nginx)

IP sor.	IP dest.	Porta sor.	Porta dest.	Prot.	Azione
172.20.1.150	172.20.1.152	*	*	*	CONSENTI
172.20.1.152	172.20.1.150	*	*	*	CONSENTI
	*	*	*	*	NEGA

Infine, analizziamo il contenuto del file di log del Security Controller (Listato A.16).

Nella prima riga vediamo la notifica di ricezione di un evento, la quale viene gestita dal metodo `public ResponseEntity<NFV> registerEvent(@RequestBody InfrastructureEvent event)`. A riga 4 e anche a riga 6 vediamo la notifica la quale afferma di non aver trovato alcun Astrid DB configurato (in quanto nel file *Astrid_ components.xml* non c'è alcun database Neo4j configurato). Nella riga 5 viene riportata la notifica del tipo di evento ricevuto (add, in questo caso), mentre dalla riga 7 inizia la fase di richiesta informazioni all'ASTRID Context Broker al fine di costruire l'*infrastructure-info* (fasi già spiegate durante l'analisi del file di log illustrato nella sezione precedente).

Si è riportato nel Listato A.17 l'ultima parte del file di log, ma solamente per completezza di informazione in quanto in essa sono presenti le notifiche generate dalle medesime operazioni che sono svolte anche nel test effettuato nella prima sezione del presente capitolo. E' utile solamente notare che, in questo caso, quando viene effettuata la verifica di esistenza dei firewall (riga 41, 55 e 61) essa da' esito positivo in quanto i firewall per le istanze dei servizi già presenti prima dell'evento accaduto sono già attivi e funzionanti, quindi avviene solamente l'aggiornamento delle regole. Per il firewall da istanziare a regolazione della nuova istanza del server apache, viene effettuata la verifica della sua esistenza (notificata a riga 47), la quale ovviamente da' esito negativo, e quindi avviene la sua creazione (notifica a riga 48).

7.3 Test ricezione evento di aggiunta nuova istanza nell"infrastruttura, con database Neo4j abilitato

Come è stato illustrato nel capitolo 5.5, esiste la possibilità di salvare gli eventi ricevuti all'interno di un database Neo4j. Per effettuare tale operazione è necessario modificare il file di configurazione raffigurato nel Listato 7.1, inserendo un nuovo componente: l'Astrid_DB. Il nuovo file di configurazione, sarà quindi il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <component name='Verefoo' IPAddress='localhost' Port='8085'/>
    <component name='ContextBroker' IPAddress='130.251.17.128' Port='5000'/>
    <component name='Astrid_DB' IPAddress='localhost' Port='7687' Username='neo4j
↪ ' Password='astrid'/>
</components>
```

Listato 7.6. contenuto del file Astrid-components.xml modificato per abilitare il salvataggio degli eventi all'interno del database Neo4j

Una volta effettuata questa modifica, rieseguiamo l'ASTRID Security Controller e inviamo nuovamente l'evento utilizzato nella sezione precedente. Controlliamo, una volta avuto risposta dal Security Controller, il contenuto del databse Neo4j, richiedendo di mostrarci tutti i dati presenti. La riposta è raffigurata nella Figura 7.3.

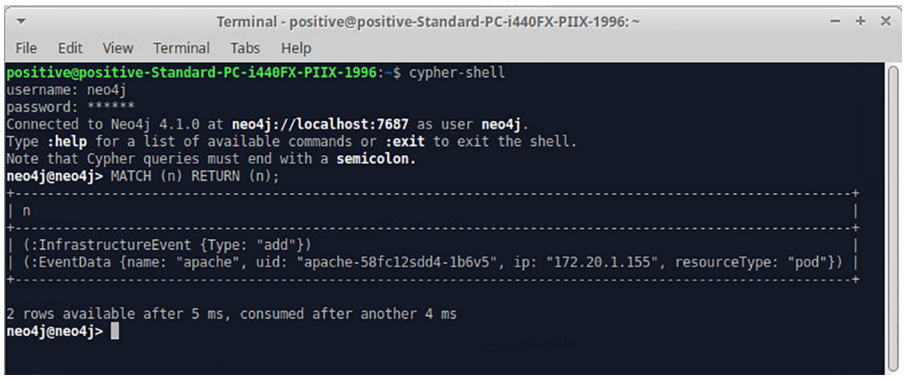


Figura 7.3. Contenuto del database Neo4j dopo la ricezione di un evento di aggiunta istanza

Dalla risposta fornita si nota che l'evento è stato correttamente inserito all'interno del database Neo4j.

Il file di log è pressochè simile a quanto già riportato nei Listati A.16 e A.17, con l'unica differenza che ora viene notificato anche la corretta aggiunta dell'evento nel database Neo4j, come si può vedere dalla sezione di tale file qui riportata:

```
2020-11-17 10:07:07.357 DEBUG 8181 --- [http-nio-8083-exec-2] o.s.web.servlet.
↪ DispatcherServlet        : POST "/controller/register/event", parameters={}
2020-11-17 10:07:07.391 DEBUG 8181 --- [http-nio-8083-exec-2] s.w.s.m.m.a.
↪ RequestMappingHandlerMapping : Mapped to public org.springframework.http.
↪ ResponseEntity<it.polito.verefoo.jaxb.NFV> it.polito.astrid.controllers.
↪ RegistrationController.registerEvent(it.polito.verefoo.astrid.jaxb.
↪ InfrastructureEvent) throws it.polito.astrid.controllers.
↪ ResourceNotFoundException,it.polito.astrid.controllers.
↪ AstridComponentNotFoundException
2020-11-17 10:07:07.545 DEBUG 8181 --- [http-nio-8083-exec-2] m.m.a.
↪ RequestResponseBodyMethodProcessor : Read "application/xml;charset=UTF-8" to [
↪ it.polito.verefoo.astrid.jaxb.InfrastructureEvent11a1b237]
2020-11-17 10:07:08.259  INFO 8181 --- [http-nio-8083-exec-2] i.p.astrid.service.
↪ RegisterEventService  : +++++++++ registerEvent Controller Obtained Event of
↪ type: add
2020-11-17 10:07:08.729  INFO 8181 --- [http-nio-8083-exec-2] Driver
↪                                  : Direct driver instance 464700566 created
↪ for server address localhost:7687
2020-11-17 10:07:12.638  INFO 8181 --- [http-nio-8083-exec-2] i.p.astrid.service.
↪ RegisterEventService  : +++++++++ Event correctly insert into Astrid DB
2020-11-17 10:07:12.644  INFO 8181 --- [http-nio-8083-exec-2] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ ask to Context Broker the infrastructure
↪  info of graph: test-network
```

Listato 7.7. File di log dell'ASTRID Security Controller dopo aver inviato un evento di aggiunta istanza con database Neo4j attivo

7.4 Test ricezione evento di rimozione di un'istanza dall'infrastruttura

Effettuiamo ora l'ultimo test che coinvolge l'ASTRID Security Controller ed il Context Broker, che riguarda la ricezione di un evento di rimozione di una istanza. Nel nostro test simuliamo la ricezione di un evento di rimozione della prima istanza di apache (non quella che è stata aggiunta nella sezione precedente). Per fare questo, è necessario simulare le azioni che sarebbero eseguite dalla dashboard di ASTRID, ovvero l'eliminazione della connessione dell'istanza da eliminare e l'eliminazione dell'*exec-env*. Iniziamo con eliminare dal Context Broker la *connection* con id pari a *apache-25fhjb7w63-8rb1s_connection*, tramite una richiesta REST di tipo DELETE all'URL *http://130.251.17.128:5000/connection/apache-25fhjb7w63-8rb1s_connection*. Fatto ciò, non ci resta che eliminare anche l'*exec-env*, inviando una richiesta DELETE all'URL *http://130.251.17.128:5000/exec-env/apache-25fhjb7w63-8rb1s*.

L'evento che inviamo all'ASTRID Security controller sull'endpoint */register/event* è il seguente:

```
<InfrastructureEvent type="delete" graphName="test-network">
    <EventData resourceType="pod" name="apache" ip="172.20.1.147" uid="apache-25
↪ fhjb7w63-8rb1s"/>
    <EventTime>2020-11-17T14:10:10.546188824Z</EventTime>
</InfrastructureEvent>
```

Listato 7.8. Dettaglio evento di rimozione istanza inviato all'ASTRID Security Controller

La risposta che ci viene fornita è riportata nel Listato A.18 dell'appendice tecnica, da cui possiamo vedere che Verefoo richiede l'istanziamento di 3 firewall per garantire le policy di raggiungibilità imposte, con le configurazioni seguenti:

- Per il servizio apache (istanza con indirizzo IP 172.20.1.155), le configurazioni sono le medesime riportate nella Tabella 7.5;
- Per il servizio nginx (istanza con indirizzo IP 172.20.1.152), le configurazioni sono riporta nella Tabella 7.7;
- Per il servizio mysql (istanza con indirizzo IP 172.20.1.170), le configurazioni sono riportate nella Tabella 7.6;

Come fatto fino ad ora, per verificare che tutto sia avvenuto correttamente, andiamo a richiedere all'ASTRID Context Broker l'elenco di tutti gli ASTRID Agents, inviando una richiesta GET all'URL *http://130.251.17.128:5000/instance/agent*, la cui risposta, che ci conferma la correttezza di tutte le operazioni, è riportata nel Listato A.19.

Riportiamo qui di seguito una sezioni significativa del file di log, diversa dalle precedenti, solamente per verificare che anche in questo caso siano avvenute correttamente tutte le notifiche:

```
2020-11-17 14:13:42.017 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet        : POST "/controller/register/event", parameters={}
2020-11-17 14:13:42.057 DEBUG 27807 --- [http-nio-8083-exec-1] s.w.s.m.m.a.
↪ RequestMappingHandlerMapping : Mapped to public org.springframework.http.
↪ ResponseEntity<it.polito.verefoo.jaxb.NFV> it.polito.astrid.controllers.
↪ RegistrationController.registerEvent(it.polito.verefoo.astrid.jaxb.
↪ InfrastructureEvent) throws it.polito.astrid.controllers.
↪ ResourceNotFoundException,it.polito.astrid.controllers.
↪ AstridComponentNotFoundException
2020-11-17 14:13:42.419 DEBUG 27807 --- [http-nio-8083-exec-1] m.m.a.
↪ RequestResponseBodyMethodProcessor : Read "application/xml;charset=UTF-8" to [
↪ it.polito.verefoo.astrid.jaxb.InfrastructureEvent1275a6fc]
2020-11-17 14:13:43.681  INFO 27807 --- [http-nio-8083-exec-1] i.p.astrid.service
↪ .RegisterEventService  : ++++++++++ registerEvent Controller Obtained Event of
↪ type: delete
2020-11-17 14:13:44.399  INFO 27807 --- [http-nio-8083-exec-1] Driver
↪                                : Direct driver instance 105151229 created
↪ for server address localhost:7687
2020-11-17 14:13:46.975  INFO 27807 --- [http-nio-8083-exec-1] i.p.astrid.service
↪ .RegisterEventService  : ++++++++++ Event correctly insert into Astrid DB
2020-11-17 14:13:47.024  INFO 27807 --- [http-nio-8083-exec-1] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ ask to Context Broker the infrastructure
↪  info of graph: test-network
2020-11-17 14:13:47.329 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate             : HTTP GET http://130.251.17.128:5000/connection
[...]
```

```
10 2020-11-17 14:14:02.773 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : HTTP DELETE http://130.251.17.128:5000/instance/
   ↪ agent/firewallapache-25fhjb7w63-8rb1s
11 2020-11-17 14:14:02.820 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Response 205 RESET_CONTENT
12 2020-11-17 14:14:02.821  INFO 27807 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ RemoveFirewallsService     : +++++++++ Firewall with ID: firewallapache-25
   ↪ fhjb7w63-8rb1s successfully deleted!
13 2020-11-17 14:14:02.877  INFO 27807 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ started to instance the firewalls
14 2020-11-17 14:14:02.918 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/instance/agent
15 2020-11-17 14:14:02.919 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Accept=[text/plain, */*]
16 2020-11-17 14:14:02.936 DEBUG 27807 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Response 200 OK
```

Listato 7.9. File di log dell'ASTRID Security Controller dopo aver inviato un evento di rimozione istanza

Possiamo notare che a riga 4 viene notificato l'evento di ricezione (correttamente di tipo *delete*) e che non viene inserito all'interno del database Neo4j in quanto esso non è configurato (tale test verrà effettuato nella successiva sezione). Nelle righe successive partono le richieste di informazioni all'ASTRID Context Broker per la costruzione dell'*infrastructure-info* (come già più volte illustrato). Da notare, infine, a riga 10 l'invio della richiesta di eliminazione del firewall riferita all'istanza che è stata rimossa.

7.5 Test ricezione evento di rimozione di un'istanza dall'infrastruttura, con database Neo4j abilitato

In questa sezione, ripetiamo le medesime operazioni svolte nella sezione precedente, ma il file *Astrid-components.xml* è sostituito dalla versione rappresentata nel Listato 7.6, in modo da abilitare il database Neo4j per la memorizzazione degli eventi ricevuti. Una volta eliminata la connessione riferita all'*exec-env* oggetto di rimozione, e l'*exec-env* stesso, rieseguiamo il Security Controller e inviamogli nuovamente l'evento rappresentato nel Listato 7.8. Osserviamo il contenuto del database Neo4j, rappresentato nella Figura 7.4.

Come confermato dallo screenshot precedente, il database Neo4j contiene ora i 2 eventi che sono stati notificati all'ASTRID Security Controller.

7.6 Test ricezione notifica di un attacco DoS

Andremo ora a testare il comportamento del Security Controller quando legge sul topic 'testing-result' una notifica di un attacco del tipo DoS. I messaggi che andremo ad inviare sul topic avranno il seguente formato:

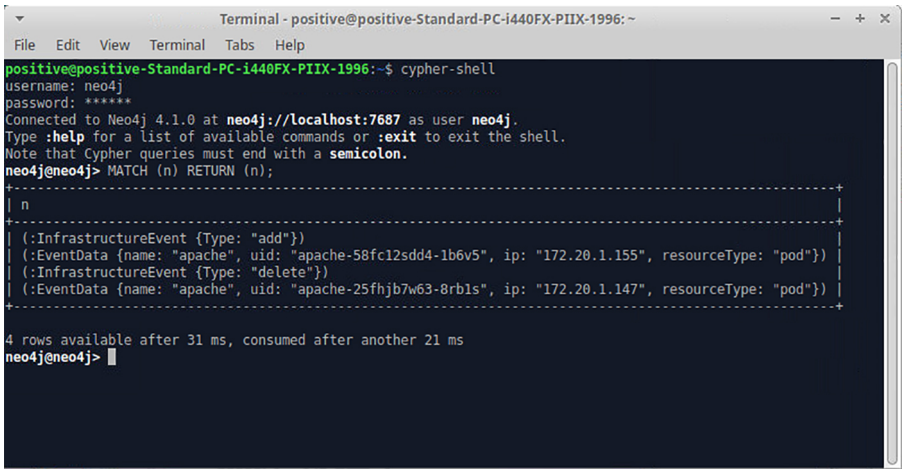


Figura 7.4. Contenuto del database Neo4j dopo la ricezione di un evento di aggiunta e uno di rimozione istanza

```
{
    "SOURCE_IP": "172.20.1.5",
    "SOURCE_PORT": 55105,
    "DESTINATION_IP": "172.20.1.150",
    "DESTINATION_PORT": 80,
    "PROTOCOL": "tcp",
    "TIMESTAMP": 1605618710.3542113,
    "ATTACK": "DDoS LOIC"
}
```

Listato 7.10. Struttura dei messaggi che andremo ad inviare sul Kafka bus

Da tale messaggio capiamo che la vittima del nostro attacco è l'istanza del database MySql. Lanciamo quindi un primo messaggio sul topic Kafka, e fino al 4° messaggio notiamo che il Security Controller notifica solamente la ricezione di un messaggio di attacco, ma non esegue alcuna azione. Nel momento in cui inviamo il 5° messaggio, dopo la notifica del messaggio d'attacco ricevuto viene anche eseguita l'azione di contromisura, la quale termina con successo. L'azione che si aspettiamo che sia stata eseguita è l'aggiornamento dell'istanza del firewall relativa al database MySql, dove viene inserita una regola aggiuntiva che va' a bloccare il traffico generato dall'IP 172.20.1.5 e destinato all'IP 172.20.1.150 (ovvero l'IP del database MySql). Richiediamo quindi al Context Broker l'elenco di tutti gli Agent Instance presenti, attravero una richiesta di tipo GET all'endpoint */instance/agent*, la cui risposta è riportata nel Listato A.21. Tale risposta ci conferma la correttezza delle operazioni svolte dal Security Controller.

7.7 Analisi delle performance delle regole Drools

Per permettere di variare facilmente le azioni svolte dall'ASTRID Security Controller, sono state introdotte le regole Drools così come è stato illustrato nel capitolo 5.4. E' importante, però, analizzare anche come questa introduzione possa influire sulle performance del Security Controller, soprattutto verificare se avviene un peggioramento o meno delle stesse all'aumentare delle regole imposte. In questa sezione andremo quindi a verificare il tempo impiegato dall'ASTRID Security Controller per analizzare le regole imposte e per decidere, a seguito di una richiesta ricevuta, quale regole è la corrispondente. Inizieremo la nostra prova con un numero molto limitato di regole (4) e andremo poi gradualmente ad aumentarle per verificare il tempo medio impiegato. Per ogni caso di test sono stati effettuate 5 ripetizioni, ovvero l'invio di 5 richieste. La Tabella 7.8 riassume i risultati ottenuti dalle prove effettuate. Analizzando i risultati, possiamo affermare che la crescita del tempo medio è molto lenta (almeno per i casi in cui le regole imposte non sono così elevate, inferiori a 500/1.000), in quanto a fronte di un aumento di 496 regole (siamo passati da 4 regole a 500 nel caso di test 5), il tempo medio è passato da 0,082s a 0,138s con un incremento percentuale quindi pari al 68% circa, ma l'aumento del numero delle regole è stato del 12.400%! Significa quindi che a fronte di una crescita di 120 volte il numero di regole, il tempo medio è cresciuto di poco più della metà. Le cose poi peggiorano se aumentiamo ancora di più il numero di regole, in quanto se passiamo da 1.000 regole a 5.000 il tempo medio aumenta di ben 0,405s, abbiamo quindi un aumento di circa il 215% del tempo medio a fronte di un aumento del 400% sul numero di regole. Infine, passando da 5.000 regole a 10.000 regole (con un incremento quindi del 100%), il tempo medio subisce un incremento del 120%, passando da 0,593s a ben 1,306s. Osservando meglio i dati ottenuti, però, balza subito all'occhio che ciò che influisce maggiormente sul tempo medio e che causa le crescite elevate è sempre la prima ripetizione di ogni caso di test, mentre le successive hanno si una crescita, ma di molto inferiore. Il motivo di questo tempo così elevato alla prima ripetizione è dovuto all'elaborazione della tavola della verità da parte del motore delle regole Drools; tale elaborazione che viene eseguita solo la prima volta che si riceve una richiesta, esegue la lettura della tavola della verità per ricavarne le regole da utilizzare poi per tutte le successive richieste. Nella Figura 7.5 riportiamo quindi i tempi di elaborazione dalla seconda ripetizione in avanti, dove si può notare più facilmente che i tempi crescono con una velocità maggiore quando il numero di regole è assai elevato (superiore a 5.000); tale affermazione è più evidente osservando la Figura 7.6 dove viene riportato l'aumento del tempo medio al crescere della variazione del numero di regole.

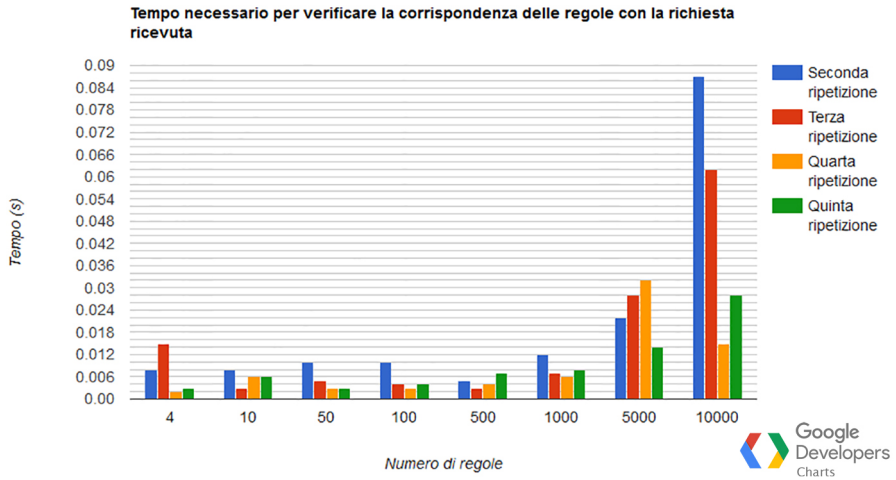


Figura 7.5. Tempi necessari all'individuazione della regola da eseguire in seguito alla ricezione di una richiesta

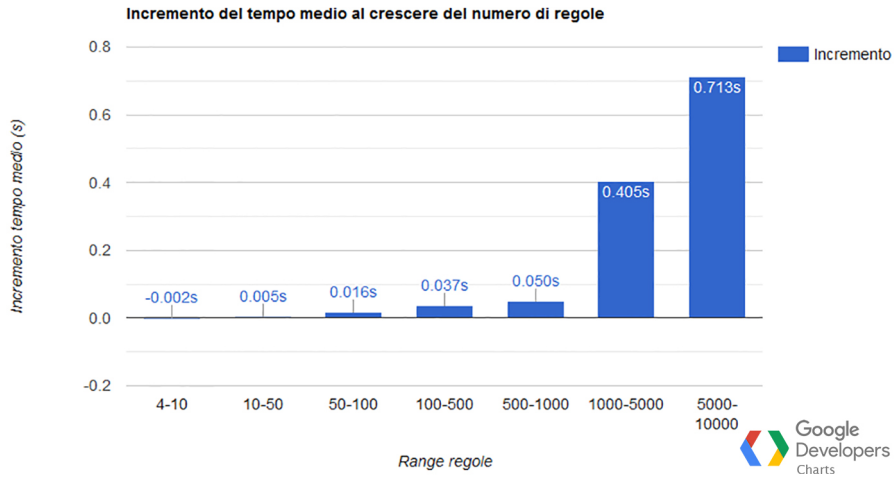


Figura 7.6. Variazione del tempo medio di individuazione della regola da eseguire, al crescere del numero di regole

Tabella 7.8: Test di analisi del tempo necessario ad individuare l'esatta regola Drools all'aumentare delle stesse

Test Case	**# Regole**	**Ripetizione**	**Ora di lettura del corpo della richiesta**	**Ora in cui viene eseguita la prima azione della regola corrispondente**	**Differenza**	**Tempo medio**
1	4	1	10:26:53,352	10:26:53,736	0,384	0,082
		2	10:27:46,364	10:27:46,372	0,008	
		3	10:29:00,456	10:29:00,471	0,015	
		4	10:29:47,159	10:29:47,161	0,002	
		5	10:30:32,725	10:30:32,728	0,003	
2	10	1	10:40:21,535	10:40:21,911	0,376	0,080
		2	10:42:04,829	10:42:04,837	0,008	
		3	10:42:34,354	10:42:34,357	0,003	
		4	10:43:02,403	10:43:02,409	0,006	
		5	10:43:37,934	10:43:37,940	0,006	
3	50	1	12:58:51,659	12:58:52,065	0,406	0,085
		2	12:59:53,855	12:59:53,865	0,010	
		3	13:00:50,494	13:00:50,499	0,005	
		4	13:01:38,382	13:01:38,385	0,003	
		5	13:02:07,090	13:02:07,093	0,003	

4	100	1	13:34:52,299	13:34:52,785	0,486	0,101
		2	13:35:35,710	13:35:35,720	0,010	
		3	13:36:09,961	13:36:09,965	0,004	
		4	13:36:42,351	13:36:42,354	0,003	
		5	13:37:12,751	13:37:12,755	0,004	
5	500	1	13:41:54,719	13:41:55,389	0,670	0,138
		2	13:42:41,063	13:42:41,068	0,005	
		3	13:43:20,471	13:43:20,474	0,003	
		4	13:43:58,706	13:43:58,710	0,004	
		5	13:44:28,497	13:44:28,504	0,007	
6	1.000	1	13:47:46,124	13:47:47,032	0,908	0,188
		2	13:48:30,790	13:48:30,802	0,012	
		3	13:49:02,127	13:49:02,134	0,007	
		4	13:49:34,972	13:49:34,978	0,006	
		5	13:50:10,187	13:50:10,195	0,008	
7	5.000	1	13:56:14,710	13:56:17,578	2,868	0,593
		2	13:57:15,849	13:57:15,871	0,022	
		3	13:58:04,903	13:58:04,931	0,028	
		4	13:58:45,195	13:58:45,227	0,032	
		5	13:59:17,929	13:59:17,943	0,014	
8	10.000	1	14:24:56,508	14:25:02,847	6,339	1,306
		2	14:28:30,404	14:28:30,491	0,087	
		3	14:29:38,738	14:29:38,800	0,062	
		4	14:30:29,437	14:30:29,452	0,015	
		5	14:31:30,244	14:31:30,272	0,028	

Capitolo 8

Conclusioni e sviluppi futuri

L'obiettivo di questa tesi sono stati la progettazione e lo sviluppo iniziale della seconda versione del Security Controller, a partire da una prima versione già sviluppata. I requisiti fondamentali della soluzione richiedevano che il comportamento del Security Controller non fosse deciso a priori, senza possibilità di modifiche nel tempo, ma era fondamentale permettere di variare tale comportamento facilmente, anche ad opera di personale non esperto. Per questo scopo si è scelto di utilizzare le regole Drools, in particolare sfruttando la tavola della verità, la quale permette di leggere e modificare le varie regole in modo meno complesso, semplicemente modificando un file di lavoro Excel.

Nella versione sviluppata durante questa tesi, il Security Controller è reso in grado di comunicare con un paio di altri moduli dell'architettura Astrid, quali il modulo Verefoo ed il Context Broker. Grazie a queste integrazioni che si sono sviluppate, il Secuirty Controller è in grado di gestire le seguenti azioni:

- Può calcolare (grazie al modulo Verefoo) ed applicare le politiche di raggiungibilità che un Network Manager impone su una certa infrastruttura di rete;
- Reagisce ad un evento di aggiunta di un'istanza di un certo servizio all'interno dell'infrastruttura di rete, ricalcolando l'allocazione ottima dei firewall ed applicandola, in modo da far rispettare comunque le regole di raggiungibilità imposte dal gestore della rete;
- Reagisce ad un evento di eliminazione di un'istanza di un certo servizio all'interno dell'infrastruttura di rete, ricalcolando l'allocazione ottima dei firewall ed applicandola, in modo da far rispettare comunque le regole di raggiungibilità imposte dal gestore della rete;
- E' in ascolto su un topic Kafka sul quale, quando legge la notifica di un attacco di tipo DoS, calcola e applica automaticamente le contromisure necessarie per bloccare tale attacco.

E' ovvio che ciò che è stato fatto fino ad ora non è sufficiente per dichiarare concluso lo sviluppo di un orchestratore di sicurezza completo. Moltissimi sono ancora gli eventi che è ancora necessario gestire, ma grazie alla basa creata con il lavoro di questa tesi, lo sviluppo successivo pensiamo possa procedere più speditamente.

In un prossimo futuro sarà necessario procedere con lo sviluppo, in particolare per permettere al Security Controller di reagire in modo automatico a diversi tipi di attacchi che può subire un'infrastruttura di rete; sarà anche necessario continuare l'integrazione con gli altri moduli dell'architettura ASTRID, mano a mano che il loro sviluppo procede.

Bibliografia

[1] VMware Carbon Black - Italian Threat Report, Giugno 2020, https://d110erj175o600.cloudfront.net/wp-content/uploads/2020/07/VMware-Carbon-Black-ItalianThreatReport.pdf

[2] Astrid project, https://www.astrid-project.eu/project.html#AnConcept

[3] M.Repetto, A.Carrega, J. Yusupov, F. Valenza, F. Risso, G. Lamanna – "Automated Security Management for Virtual Services" in 2019 IEEE Conference on Network Function Vitualization and Software Defined Network (NFV – SDN)

[4] OSM White Paper - Issue 1, February 2019, https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf

[5] OpenStack Tacker Wiki, https://wiki.openstack.org/wiki/Tacker

[6] OpenStack Tacker Documentazione, https://docs.openstack.org/heat/latest/

[7] Open Baton Documentazione, https://openbaton-docs.readthedocs.io/en/3.0.0/

[8] Cisco documentazione ESC, https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/elastic-services-controller-esc/datasheet-c78-734670.html

[9] Cisco documentazione NSO, https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/datasheet-c78-734576.html

[10] Ericsson Orchestrator documentazione, https://www.ericsson.com/en/portfolio/digital-services/automated-network-operations/orchestration/ericsson-orchestrator

[11] ZTE CloudStudio documentazione, https://sdnfv.zte.com.cn/en/products/MANO

[12] Introduzione al Framework Spring - italianCoders, https://italiancoders.it/introduzione-spring-inversion-of-control/

[13] Cos'è la Dependency Injection - devexp.io, https://bit.ly/37nz85L

[14] L'architettura di Spring - javaboss, https://www.javaboss.it/architettura-spring/

[15] Drools Introduction - tutorialspoint.com, https://www.tutorialspoint.com/drools/drools_introduction.htm

[16] Introduction to Drools - baeldung.com, https://www.baeldung.com/drools
[17] Drools rules engine tutorial - mastertheboss.com, http://www.mastertheboss.com/jboss-jbpm/drools/jboss-drools-tutorial
[18] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, J. Yusupov – "Introducing programmability and automation in the synthesis of virtual firewall rules"
[19] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, J. Yusupov – Appunti sulla seconda versione dell'ASTRID Security Controller
[20] ASTRID Context Broker Manager - https://github.com/astrid-project/cb-manager
[21] Neo4j - Guide per gli sviluppatori, https://neo4j.com/developer/graph-database/
[22] Apache Kafka - https://kafka.apache.org/documentation

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Appendice A

Appendice Tecnica

Tutti gli aspetti più tecnici, citati nel corso della trattazione della tesi, come le risposte fornite dai vari moduli, le query d'interrogazione e altro, sono riportati all'interno di questa appendice, in modo da alleggerire il contenuto della tesi stessa ma permettere comunque un maggior approfondimento a chi è maggiormente interessato.

```
import it.polito.astrid.service.*;
import it.polito.astrid.models.*;
import it.polito.verefoo.astrid.jaxb.Components.*;
import it.polito.verefoo.astrid.jaxb.InfrastructureEvent.*;
import org.springframework.http.ResponseEntity;

rule "Register infrastructure info"
    when
        interceptObject: InterceptionRequest(command=="/register/insfrastructure"
↪ )
        component: Component()
    then
        RegisterInfrastructureService IS = new RegisterInfrastructureService();
        interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo
↪ (), component));
    end

rule "Register Policy"
    when
        interceptObject: InterceptionRequest(command=="/register/policy")
        component1: Component(name=="Verefoo")
        component2: Component(name=="ContextBroker")
    then
        RegisterPolicyService RP = new RegisterPolicyService();
        ResponseEntity<String> res = RP.registerPolicy(interceptObject.getPolicy
↪ (), component1);
        InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(res,
↪ component2, component1);
        interceptObject.setInfo(IIR.getInfrastructureInfo());
        RegisterInfrastructureService IS = new RegisterInfrastructureService();
        interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo
↪ (), component1));
        InstanceFirewallsService IFS = new InstanceFirewallsService(component2,
↪ interceptObject.getNfv().getBody());
        IFS.instanceFirewallsFromNFV();
```

Listato A.1. File XML che rappresenta la configurazione dei firewall fornita dal modulo Verekube al Security Controller, che la fornisce in output

```
<NFV>
    <graphs>
        <Graph id="55">
            <node name="2.208.56.74" functional_type="WEBSERVER">
                <neighbour name="2.208.56.31"/>
                <configuration name="Apache" description="8080">
                    <webserver>
                        <Name>2.208.56.74</Name>
                    </webserver>
                </configuration>
            </node>
            <node name="2.208.56.31" functional_type="FIREWALL">
                <neighbour name="2.208.56.74"/>
                <neighbour name="2.208.56.32"/>
                <neighbour name="2.208.56.33"/>
                <neighbour name="2.208.56.34"/>
                <configuration name="apache-697b84bd87-b4rkz" description="
↪ 2.208.56.74">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>2.208.56.-1</source>
                            <destination>2.208.56.-1</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
                </configuration>
            </node>
            <node name="2.208.56.75" functional_type="WEBSERVER">
                <neighbour name="2.208.56.32"/>
                <configuration name="Neo4j_1" description="7687">
                    <webserver>
                        <Name>2.208.56.75</Name>
                    </webserver>
                </configuration>
            </node>
            <node name="2.208.56.32" functional_type="FIREWALL">
                <neighbour name="2.208.56.75"/>
                <neighbour name="2.208.56.31"/>
                <neighbour name="2.208.56.33"/>
                <neighbour name="2.208.56.34"/>
                <configuration name="neo4j-55c9447f74-wn5c4" description="
↪ 2.208.56.75">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>2.208.56.-1</source>
                            <destination>2.208.56.-1</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
                </configuration>
            </node>
```

```
            <node name="2.208.56.76" functional_type="WEBSERVER">
                <neighbour name="2.208.56.33"/>
                <configuration name="Neo4j_2" description="7687">
                    <webserver>
                        <Name>2.208.56.76</Name>
                    </webserver>
                </configuration>
            </node>
            <node name="2.208.56.33" functional_type="FIREWALL">
                <neighbour name="2.208.56.76"/>
                <neighbour name="2.208.56.31"/>
                <neighbour name="2.208.56.32"/>
                <neighbour name="2.208.56.34"/>
                <configuration name="neo4j-84f7f9f55c-8f7zj" description="
↪ 2.208.56.76">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>2.208.56.-1</source>
                            <destination>2.208.56.-1</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
                </configuration>
            </node>
            <node name="2.208.56.77" functional_type="WEBSERVER">
                <neighbour name="2.208.56.34"/>
                <configuration name="Ngix" description="8080">
                    <webserver>
                        <Name>2.208.56.77</Name>
                    </webserver>
                </configuration>
            </node>
            <node name="2.208.56.34" functional_type="FIREWALL">
                <neighbour name="2.208.56.77"/>
                <neighbour name="2.208.56.31"/>
                <neighbour name="2.208.56.32"/>
                <neighbour name="2.208.56.33"/>
                <configuration name="ngix-v4y83n462c-c2d9l" description="
↪ 2.208.56.77">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>2.208.56.77</source>
                            <destination>2.208.56.74</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                        <elements>
                            <action>ALLOW</action>
                            <source>2.208.56.74</source>
                            <destination>2.208.56.77</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
```

```
                </configuration>
            </node>
        </Graph>
    </graphs>
    <Constraints>
        <NodeConstraints/>
        <LinkConstraints/>
    </Constraints>
    <PropertyDefinition>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.74" dst="
↪ 2.208.56.75" Src_port="8080" Dst_port="7687" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.74" dst="
↪ 2.208.56.76" Src_port="8080" Dst_port="7687" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.74" dst="
↪ 2.208.56.77" Src_port="8080" Dst_port="8080" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.75" dst="
↪ 2.208.56.76" Src_port="7687" Dst_port="7687" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.76" dst="
↪ 2.208.56.75" Src_port="7687" Dst_port="7687" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.75" dst="
↪ 2.208.56.74" Src_port="7687" Dst_port="8080" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.76" dst="
↪ 2.208.56.74" Src_port="7687" Dst_port="8080" isSat="true"/>
        <Property name="ReachabilityProperty" graph="55" src="2.208.56.77" dst="
↪ 2.208.56.74" Src_port="8080" Dst_port="8080" isSat="true"/>
    </PropertyDefinition>
</NFV>
```

```
    end

rule "Register Event - Add a Instance"
    when
        interceptObject: InterceptionRequest(command=="/register/event",
                         event.type=="add")
        component1: Component(name=="Verefoo")
        component2: Component(name=="Astrid_DB")
        component3: Component(name=="ContextBroker")
    then
        RegisterEventService REES = new RegisterEventService();
        String graphName = REES.registerEvent(interceptObject.getEvent(),
↪ component2);
        InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(component3)
↪ ;
        interceptObject.setInfo(IIR.getInfrastructureInfo(graphName));
        RegisterInfrastructureService IS = new RegisterInfrastructureService();
        interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo
↪ (), component1));
        InstanceFirewallsService IFS = new InstanceFirewallsService(component3,
↪ interceptObject.getNfv().getBody());
        IFS.instanceFirewallsFromNFV();
    end

rule "Register Event - Remove a Instance"
    when
        interceptObject: InterceptionRequest(command=="/register/event",
                         event.type=="delete")
        component1: Component(name=="Verefoo")
        component2: Component(name=="Astrid_DB")
        component3: Component(name=="ContextBroker")
    then
        RegisterEventService REES = new RegisterEventService();
        String graphName = REES.registerEvent(interceptObject.getEvent(),
↪ component2);
        InfrastructureInfoRequest IIR = new InfrastructureInfoRequest(component3)
↪ ;
        interceptObject.setInfo(IIR.getInfrastructureInfo(graphName));
        RegisterInfrastructureService IS = new RegisterInfrastructureService();
        interceptObject.setNfv(IS.registerInfrastructure(interceptObject.getInfo
↪ (), component1));
        RemoveFirewallsService RFS = new RemoveFirewallsService(component3);
        RFS.removeFirewallfromInfrastructureEvent(interceptObject.getEvent());
        InstanceFirewallsService IFS = new InstanceFirewallsService(component3,
↪ interceptObject.getNfv().getBody());
        IFS.instanceFirewallsFromNFV();
    end
```

Listato A.2. Regole Drools

```
[
    {
        "description": "apache-55c9447f74-wn5c4 network connection",
        "exec_env_id": "apache-55c9447f74-wn5c4",
        "id": "apache-55c9447f74-wn5c4_connection",
        "network_link_id": "my-graph"
    },
    {
        "description": "nodejs-84f7f9f55c-8f7zj network connection",
        "exec_env_id": "nodejs-84f7f9f55c-8f7zj",
        "id": "nodejs-84f7f9f55c-8f7zj_connection",
        "network_link_id": "my-graph"
```

```
    },
    {
        "description": "nginx-697b84bd87-b4rkz network connection",
        "exec_env_id": "nginx-697b84bd87-b4rkz",
        "id": "nginx-697b84bd87-b4rkz_connection",
        "network_link_id": "my-graph"
    },
    {
        "description": "apache-55c9447f74-wn5c7 network connection",
        "exec_env_id": "apache-55c9447f74-wn5c7",
        "id": "apache-55c9447f74-wn5c7_connection",
        "network_link_id": "another-graph"
    }
]
```

Listato A.3. Esempio di risposta fornita dal Context Broker a seguito di una richiesta all'endpoint *connection*

```
{
    "config": {
        "cmd": "polycubectl firewall add fw"
    },
    "id": "start",
    "status": "started"
},
{
    "config": {
        "cmd": "polycubectl fw del"
    },
    "id": "stop",
    "status": "stopped"
},
{
    "config": {
        "cmd": "polycubectl attach fw {port}"
    },
    "id": "attach"
},
{
    "config": {
        "cmd": "polycubectl firewall fw chain {chain} insert id={n} src={src} dst
↪ ={dst} action={action}"
    },
    "id": "insert"
},
{
    "config": {
        "cmd": "polycubectl firewall fw chain {chain} append src={src} dst={dst}
↪ action={action}"
    },
    "id": "append"
},
{
    "config": {
        "cmd": "polycubectl firewall fw chain {chain} insert src={src} dst={dst}
↪ action={action}"
    },
    "id": "prepend"
},
{
    "config": {
        "cmd": "polycubectl firewall fw chain {chain} rule del {n}"
    },
```

```
43      "id": "delete"
44  },
45  {
46      "config": {
47          "cmd": "polycubectl firewall fw chain {chain} set default={action}"
48      },
49      "id": "default"
50  },
51  {
52      "config": {
53          "cmd": "polycubectl firewall fw chain {chain} rule show"
54      },
55      "id": "list"
56  },
57  {
58      "config": {
59          "cmd": "polycubectl firewall fw chain {chain} stats show"
60      },
61      "id": "stats"
62  }
```

Listato A.4. Elenco delle azioni supportate dall'ASTRID Agent Instance firewall

```
1  {
2      "exec_env_id":"apache-55c9447f74-wn5c4",
3      "agent_catalog_id":"firewall",
4      "id":"firewall@apache-55c9447f74-wn5c4",
5      "actions":[
6      {
7          "action":"DENY",
8          "id":"default",
9          "timestamp":"2020-11-10T10:06:20"
10     },
11     {
12         "action":"ALLOW",
13         "src":"2.208.56.0/24",
14         "dst":"2.208.56.0/24",
15         "id":"insert",
16         "n":"rule1@apache-55c9447f74-wn5c4",
17         "timestamp":"2020-11-10T10:06:20"
18     },
19     {
20         "action":"ALLOW",
21         "src":"2.208.56.11",
22         "dst":"2.208.56.14",
23         "id":"insert",
24         "n":"rule2@apache-55c9447f74-wn5c4",
25         "timestamp":"2020-11-10T10:06:20"
26     }
27     ],
28     "status":"stopped"
29 }
```

Listato A.5. Esempio di query d'inserimento di un firewall all'interno dell'ASTRID Context Broker

```
1  [
2      {
3          "description": "mysql server instance",
4          "enabled": true,
5          "hostname": "172.20.1.150",
6          "id": "mysql-v4f8ss32ce-2f5v4",
7          "lcp": {
```

```
            "port": 8000
        },
        "type_id": "mysql"
    },
    {
        "description": "apache server instance",
        "enabled": true,
        "hostname": "172.20.1.147",
        "id": "apache-25fhjb7w63-8rb1s",
        "lcp": {
            "port": 80
        },
        "type_id": "apache"
    },
    {
        "description": "VM on laptop",
        "enabled": true,
        "hostname": "172.21.170.140",
        "id": "vm-laptop",
        "lcp": {
            "last_heartbeat": "2020-11-12T13:47:08",
            "password": "8c5762500f2ea58aa93b2efae710b1f4f3613fd12e",
            "port": 4000,
            "started": "2020-11-12T11:22:40",
            "username": "2cc1ea08-24e5-11eb-9e2b-3df8a2e"
        },
        "type_id": "vm"
    },
    {
        "description": "nginx server instance",
        "enabled": true,
        "hostname": "172.20.1.152",
        "id": "nginx-ed46c14d85-rf12v",
        "lcp": {
            "port": 8080
        },
        "type_id": "nginx"
    },
    {
        "description": "eBPF test",
        "enabled": true,
        "hostname": "192.168.0.172",
        "id": "ebpf_test",
        "lcp": {
            "last_heartbeat": "2020-11-12T13:15:52",
            "password": "7c33d84cc90aaf131d",
            "port": 4000,
            "started": "2020-11-10T10:36:21",
            "username": "306c269c-24e9-11eb-b9"
        },
        "type_id": "container-docker"
    }
]
```


Listato A.6. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutti gli exec-env


```
[
    {
        "description": "test2_network",
        "id": "another-graph",
        "type_id": "wifi"
    },
```

```
    {
        "description": "network of test",
        "id": "test-network",
        "type_id": "wifi"
    }
]
```

Listato A.7. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutti i network-link

```
[
    {
        "description": "apache-25fhjb7w63-8rb1s network connection",
        "exec_env_id": "apache-25fhjb7w63-8rb1s",
        "id": "apache-25fhjb7w63-8rb1s_connection",
        "network_link_id": "test-network"
    },
    {
        "description": "mysql-v4f8ss32ce-2f5v4 network connection",
        "exec_env_id": "mysql-v4f8ss32ce-2f5v4",
        "id": "mysql-v4f8ss32ce-2f5v4_connection",
        "network_link_id": "test-network"
    },
    {
        "description": "nginx-ed46c14d85-rf12v network connection",
        "exec_env_id": "nginx-ed46c14d85-rf12v",
        "id": "nginx-ed46c14d85-rf12v_connection",
        "network_link_id": "test-network"
    }
]
```

Listato A.8. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutte le connection

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NFV>
    <graphs>
        <graph id="29">
            <node name="172.20.1.147" functional_type="WEBSERVER">
                <neighbour name="172.20.1.167"/>
                <configuration name="apache" description="80">
                    <webserver>
                        <name>172.20.1.147</name>
                    </webserver>
                </configuration>
            </node>
            <node name="172.20.1.167" functional_type="FIREWALL">
                <neighbour name="172.20.1.147"/>
                <neighbour name="172.20.1.170"/>
                <neighbour name="172.20.1.172"/>
                <configuration name="apache-25fhjb7w63-8rb1s" description="
↪ 172.20.1.147">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>172.20.1.150</source>
                            <destination>172.20.1.147</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                        <elements>
```

```
                        <action>ALLOW</action>
                        <source>172.20.1.147</source>
                        <destination>172.20.1.150</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="172.20.1.150" functional_type="WEBSERVER">
            <neighbour name="172.20.1.170"/>
            <configuration name="mysql" description="8000">
                <webserver>
                    <name>172.20.1.150</name>
                </webserver>
            </configuration>
        </node>
        <node name="172.20.1.170" functional_type="FIREWALL">
            <neighbour name="172.20.1.150"/>
            <neighbour name="172.20.1.167"/>
            <neighbour name="172.20.1.172"/>
            <configuration name="mysql-v4f8ss32ce-2f5v4" description="
↪ 172.20.1.150">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.-1</source>
                        <destination>172.20.1.-1</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="172.20.1.152" functional_type="WEBSERVER">
            <neighbour name="172.20.1.172"/>
            <configuration name="nginx" description="8080">
                <webserver>
                    <name>172.20.1.152</name>
                </webserver>
            </configuration>
        </node>
        <node name="172.20.1.172" functional_type="FIREWALL">
            <neighbour name="172.20.1.152"/>
            <neighbour name="172.20.1.167"/>
            <neighbour name="172.20.1.170"/>
            <configuration name="nginx-ed46c14d85-rf12v" description="
↪ 172.20.1.152">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.150</source>
                        <destination>172.20.1.152</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.152</source>
                        <destination>172.20.1.150</destination>
```

```
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
    </graph>
</graphs>
<Constraints>
    <NodeConstraints/>
    <LinkConstraints/>
</Constraints>
<PropertyDefinition>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.147" dst="
↪ 172.20.1.150" src_port="80" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.147" src_port="8000" dst_port="80" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.152" dst="
↪ 172.20.1.150" src_port="8080" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.152" src_port="8000" dst_port="8080" isSat="true"/>
</PropertyDefinition>
</NFV>
```

Listato A.9. Risposta fornita dall'ASTRID Security Controller all'invio della richiesta sull'endpoint *register/policy*

```
[
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-16T10:51:00"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.147",
                "id": "insert",
                "n": "rule1@apache-25fhjb7w63-8rb1s",
                "src": "172.20.1.150",
                "timestamp": "2020-11-16T10:51:00"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@apache-25fhjb7w63-8rb1s",
                "src": "172.20.1.147",
                "timestamp": "2020-11-16T10:51:00"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "apache-25fhjb7w63-8rb1s",
        "id": "firewall@apache-25fhjb7w63-8rb1s",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
```

```
                "id": "default",
                "timestamp": "2020-11-16T10:51:00"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.0/24",
                "id": "insert",
                "n": "rule1@mysql-v4f8ss32ce-2f5v4",
                "src": "172.20.1.0/24",
                "timestamp": "2020-11-16T10:51:00"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "mysql-v4f8ss32ce-2f5v4",
        "id": "firewall@mysql-v4f8ss32ce-2f5v4",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-16T10:51:00"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.152",
                "id": "insert",
                "n": "rule1@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.150",
                "timestamp": "2020-11-16T10:51:00"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.152",
                "timestamp": "2020-11-16T10:51:00"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "nginx-ed46c14d85-rf12v",
        "id": "firewall@nginx-ed46c14d85-rf12v",
        "status": "started"
    }
]
```


Listato A.10. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutti gli *ASTRID Agents*

```
2020-11-16 10:50:28.211  INFO 7748 --- [main] o.s.b.w.embedded.tomcat.
↪ TomcatWebServer  : Tomcat started on port(s): 8083 (http) with context path '/
↪ controller'
2020-11-16 10:50:28.221  INFO 7748 --- [main] i.p.a.controllers.
↪ ControllerApplication  : Started ControllerApplication in 8.909 seconds (JVM
↪ running for 9.566)
2020-11-16 10:50:49.837  INFO 7748 --- [http-nio-8083-exec-1] o.a.c.c.C.[.[
↪ localhost].[/controller]    : Initializing Spring DispatcherServlet '
↪ dispatcherServlet'
2020-11-16 10:50:49.839  INFO 7748 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
```

```
2020-11-16 10:50:49.840 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet        : Detected StandardServletMultipartResolver
2020-11-16 10:50:49.884 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet        : enableLoggingRequestDetails='false': request
↪ parameters and headers will be masked to prevent unsafe logging of potentially
↪ sensitive data
2020-11-16 10:50:49.885  INFO 7748 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet        : Completed initialization in 46 ms
2020-11-16 10:50:49.906 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet        : POST "/controller/register/policy", parameters={}
2020-11-16 10:50:49.927 DEBUG 7748 --- [http-nio-8083-exec-1] s.w.s.m.m.a.
↪ RequestMappingHandlerMapping : Mapped to public org.springframework.http.
↪ ResponseEntity<it.polito.verefoo.jaxb.NFV> it.polito.astrid.controllers.
↪ RegistrationController.registerPolicy(java.lang.String) throws it.polito.astrid
↪ .controllers.AstridComponentNotFoundException,java.io.IOException,it.polito.
↪ astrid.controllers.ResourceNotFoundException
2020-11-16 10:50:49.983 DEBUG 7748 --- [http-nio-8083-exec-1] m.m.a.
↪ RequestResponseBodyMethodProcessor : Read "text/plain;charset=UTF-8" to ["
↪ metadata:
  name: test-network
spec:
  policies:
      - from: apache
        to: mysql
        act (truncated)...]
2020-11-16 10:50:50.333  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
↪ RegisterPolicyService      : ++++++++++ registerPolicy Controller Obtained
↪ Policy
2020-11-16 10:50:50.354  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
↪ RegisterPolicyService      : ++++++++++ registerPolicy Controller Sending to
↪ Verekube
2020-11-16 10:50:50.394 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : HTTP POST http://localhost:8085/verefoo/graph
2020-11-16 10:50:50.402 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, application/xml, text/xml,
↪ application/json, application/*+xml, application/*+json, */*]
2020-11-16 10:50:50.406 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Writing [metadata:
  name: test-network
spec:
  policies:
      - from: apache
        to: mysql
        action: forward
      - from: mysql
        to: apache
        action: forward
      - from: nginx
        to: mysql
        action: forward
      - from: mysql
        to: nginx
        action: forward] as "text/plain"
```

Listato A.11. File di log dell'ASTRID Security Controller - Prima parte

```
2020-11-16 10:50:50.522 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-16 10:50:50.524 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "text/plain;
↪ charset=ISO-8859-1"
```

```
39 2020-11-16 10:50:50.549  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ RegisterPolicyService       : ++++++++++ registerPolicy Controller Success from
   ↪ Verekube: Verikube Successfully added: test-network
40 2020-11-16 10:50:50.552  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest   : ++++++++++ ask to Context Broker the infrastructure
   ↪  info of graph: test-network
41 2020-11-16 10:50:50.566 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : HTTP GET http://130.251.17.128:5000/connection
42 2020-11-16 10:50:50.567 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Accept=[text/plain, */*]
43 2020-11-16 10:50:50.598 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Response 200 OK
44 2020-11-16 10:50:50.599 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Reading to [java.lang.String] as "application/json"
45 2020-11-16 10:50:50.602  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest   : ++++++++++ Context Broker answer successfully with
   ↪  connection of requested network
46 2020-11-16 10:50:50.675 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : HTTP GET http://130.251.17.128:5000/exec-env/apache
   ↪ -25fhjb7w63-8rb1s
47 2020-11-16 10:50:50.676 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Accept=[text/plain, */*]
48 2020-11-16 10:50:50.696 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Response 200 OK
49 2020-11-16 10:50:50.697 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Reading to [java.lang.String] as "application/json"
50 2020-11-16 10:50:50.713 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : HTTP GET http://130.251.17.128:5000/exec-env/mysql-
   ↪ v4f8ss32ce-2f5v4
51 2020-11-16 10:50:50.714 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Accept=[text/plain, */*]
52 2020-11-16 10:50:50.731 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Response 200 OK
53 2020-11-16 10:50:50.732 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Reading to [java.lang.String] as "application/json"
54 2020-11-16 10:50:50.736 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : HTTP GET http://130.251.17.128:5000/exec-env/nginx-
   ↪ ed46c14d85-rf12v
55 2020-11-16 10:50:50.737 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Accept=[text/plain, */*]
56 2020-11-16 10:50:50.754 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Response 200 OK
57 2020-11-16 10:50:50.756 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Reading to [java.lang.String] as "application/json"
58 2020-11-16 10:50:50.757  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.s.
   ↪ RegisterInfrastructureService     : ++++++++++ registerInfrastructure Controller
   ↪  Obtained Infrastructure info for: test-network
59 2020-11-16 10:50:50.769  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.s.
   ↪ RegisterInfrastructureService     : ++++++++++ registerInfrastructure Controller
   ↪  Sending Infrastructure info to Verekube
60 2020-11-16 10:50:50.773 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : HTTP POST http://localhost:8085/verefoo/dc
61 2020-11-16 10:50:50.774 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Accept=[*/*]
62 2020-11-16 10:50:50.774 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Writing [it.polito.verefoo.astrid.jaxb.
   ↪ InfrastructureInfo70765422] as "application/xml"
63 2020-11-16 10:51:00.537 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Response 200 OK
64 2020-11-16 10:51:00.538 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate                : Reading to [it.polito.verefoo.jaxb.NFV] as "
   ↪ application/xml"
```

Listato A.12. File di log dell'ASTRID Security Controller - Seconda parte

```
65 2020-11-16 10:51:00.703  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ started to instance the firewalls
66 2020-11-16 10:51:00.713 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/instance/agent
67 2020-11-16 10:51:00.714 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Accept=[text/plain, */*]
68 2020-11-16 10:51:00.743 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Response 200 OK
69 2020-11-16 10:51:00.743 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
70 2020-11-16 10:51:00.750  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID:
   ↪ apache-25fhjb7w63-8rb1s already exist
71 2020-11-16 10:51:00.751  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: apache-25
   ↪ fhjb7w63-8rb1s not exist. Creating in progress...
72 2020-11-16 10:51:00.757 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : HTTP POST http://130.251.17.128:5000/instance/agent
73 2020-11-16 10:51:00.758 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Accept=[text/plain, */*]
74 2020-11-16 10:51:00.758 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Writing [{"exec_env_id":"apache-25fhjb7w63-8rb1s","
   ↪ agent_catalog_id":"firewall","id":"firewall@apache-25fhjb7w63-8rb1s","actions":
   ↪ [{"action":"DENY","id":"default","timestamp":"2020-11-16T10:51:00"},{"dst":"
   ↪ 172.20.1.147","src":"172.20.1.150","action":"ALLOW","id":"insert","n":"
   ↪ rule1@apache-25fhjb7w63-8rb1s","timestamp":"2020-11-16T10:51:00"},{"dst":"
   ↪ 172.20.1.150","src":"172.20.1.147","action":"ALLOW","id":"insert","n":"
   ↪ rule2@apache-25fhjb7w63-8rb1s","timestamp":"2020-11-16T10:51:00"}],"status":"
   ↪ started"}] as "application/json"
75 2020-11-16 10:51:00.819 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Response 201 CREATED
76 2020-11-16 10:51:00.820 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
77 2020-11-16 10:51:00.820  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully instanced!
78 2020-11-16 10:51:00.821  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID: mysql
   ↪ -v4f8ss32ce-2f5v4 already exist
79 2020-11-16 10:51:00.821  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: mysql-
   ↪ v4f8ss32ce-2f5v4 not exist. Creating in progress...
80 2020-11-16 10:51:00.823 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : HTTP POST http://130.251.17.128:5000/instance/agent
81 2020-11-16 10:51:00.824 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Accept=[text/plain, */*]
82 2020-11-16 10:51:00.824 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Writing [{"exec_env_id":"mysql-v4f8ss32ce-2f5v4","
   ↪ agent_catalog_id":"firewall","id":"firewall@mysql-v4f8ss32ce-2f5v4","actions":
   ↪ [{"action":"DENY","id":"default","timestamp":"2020-11-16T10:51:00"},{"dst":"
   ↪ 172.20.1.0\/24","src":"172.20.1.0\/24","action":"ALLOW","id":"insert","n":"
   ↪ rule1@mysql-v4f8ss32ce-2f5v4","timestamp":"2020-11-16T10:51:00"}],"status":"
   ↪ started"}] as "application/json"
83 2020-11-16 10:51:00.873 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Response 201 CREATED
84 2020-11-16 10:51:00.874 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
   ↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
85 2020-11-16 10:51:00.875  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully instanced!
```

```
2020-11-16 10:51:00.875  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID: nginx
↪ -ed46c14d85-rf12v already exist
2020-11-16 10:51:00.875  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: nginx-
↪ ed46c14d85-rf12v not exist. Creating in progress...
2020-11-16 10:51:00.879 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : HTTP POST http://130.251.17.128:5000/instance/agent
2020-11-16 10:51:00.880 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
2020-11-16 10:51:00.881 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Writing [{"exec_env_id":"nginx-ed46c14d85-rf12v","
↪ agent_catalog_id":"firewall","id":"firewall@nginx-ed46c14d85-rf12v","actions":
↪ [{"action":"DENY","id":"default","timestamp":"2020-11-16T10:51:00"},{"dst":"
↪ 172.20.1.152","src":"172.20.1.150","action":"ALLOW","id":"insert","n":"
↪ rule1@nginx-ed46c14d85-rf12v","timestamp":"2020-11-16T10:51:00"},{"dst":"
↪ 172.20.1.150","src":"172.20.1.152","action":"ALLOW","id":"insert","n":"
↪ rule2@nginx-ed46c14d85-rf12v","timestamp":"2020-11-16T10:51:00"}],"status":"
↪ started"}] as "application/json"
2020-11-16 10:51:00.929 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Response 201 CREATED
2020-11-16 10:51:00.930 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-16 10:51:00.931  INFO 7748 --- [http-nio-8083-exec-1] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully instanced!
2020-11-16 10:51:00.951 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.w.s.m.m.a.
↪ HttpEntityMethodProcessor  : Found 'Content-Type:application/xml' in response
2020-11-16 10:51:00.953 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.w.s.m.m.a.
↪ HttpEntityMethodProcessor  : Writing [it.polito.verefoo.jaxb.NFV6fc78ee6]
2020-11-16 10:51:01.125 DEBUG 7748 --- [http-nio-8083-exec-1] o.s.web.servlet.
↪ DispatcherServlet          : Completed 200 OK
```

Listato A.13. File di log dell'ASTRID Security Controller - Terza parte

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NFV>
    <graphs>
        <graph id="29">
            <node name="172.20.1.147" functional_type="WEBSERVER">
                <neighbour name="172.20.1.167"/>
                <configuration name="apache" description="80">
                    <webserver>
                        <name>172.20.1.147</name>
                    </webserver>
                </configuration>
            </node>
            <node name="172.20.1.167" functional_type="FIREWALL">
                <neighbour name="172.20.1.147"/>
                <neighbour name="172.20.1.175"/>
                <neighbour name="172.20.1.170"/>
                <neighbour name="172.20.1.172"/>
                <configuration name="apache-25fhjb7w63-8rb1s" description="
↪ 172.20.1.147">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>172.20.1.150</source>
                            <destination>172.20.1.147</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                        <elements>
```

```
                        <action>ALLOW</action>
                        <source>172.20.1.147</source>
                        <destination>172.20.1.150</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="172.20.1.155" functional_type="WEBSERVER">
            <neighbour name="172.20.1.175"/>
            <configuration name="apache" description="80">
                <webserver>
                    <name>172.20.1.155</name>
                </webserver>
            </configuration>
        </node>
        <node name="172.20.1.175" functional_type="FIREWALL">
            <neighbour name="172.20.1.155"/>
            <neighbour name="172.20.1.167"/>
            <neighbour name="172.20.1.170"/>
            <neighbour name="172.20.1.172"/>
            <configuration name="apache-58fc12sdd4-1b6v5" description="
↪ 172.20.1.155">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.155</source>
                        <destination>172.20.1.150</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.150</source>
                        <destination>172.20.1.155</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="172.20.1.150" functional_type="WEBSERVER">
            <neighbour name="172.20.1.170"/>
            <configuration name="mysql" description="8000">
                <webserver>
                    <name>172.20.1.150</name>
                </webserver>
            </configuration>
        </node>
        <node name="172.20.1.170" functional_type="FIREWALL">
            <neighbour name="172.20.1.150"/>
            <neighbour name="172.20.1.167"/>
            <neighbour name="172.20.1.175"/>
            <neighbour name="172.20.1.172"/>
            <configuration name="mysql-v4f8ss32ce-2f5v4" description="
↪ 172.20.1.150">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
```

```
                        <source>172.20.1.-1</source>
                        <destination>172.20.1.-1</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="172.20.1.152" functional_type="WEBSERVER">
            <neighbour name="172.20.1.172"/>
            <configuration name="nginx" description="8080">
                <webserver>
                    <name>172.20.1.152</name>
                </webserver>
            </configuration>
        </node>
        <node name="172.20.1.172" functional_type="FIREWALL">
            <neighbour name="172.20.1.152"/>
            <neighbour name="172.20.1.167"/>
            <neighbour name="172.20.1.175"/>
            <neighbour name="172.20.1.170"/>
            <configuration name="nginx-ed46c14d85-rf12v" description="
↪ 172.20.1.152">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.150</source>
                        <destination>172.20.1.152</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.152</source>
                        <destination>172.20.1.150</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
    </graph>
</graphs>
<Constraints>
    <NodeConstraints/>
    <LinkConstraints/>
</Constraints>
<PropertyDefinition>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.147" dst="
↪ 172.20.1.150" src_port="80" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.155" dst="
↪ 172.20.1.150" src_port="80" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.147" src_port="8000" dst_port="80" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.155" src_port="8000" dst_port="80" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.152" dst="
↪ 172.20.1.150" src_port="8080" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.152" src_port="8000" dst_port="8080" isSat="true"/>
```

```
    </PropertyDefinition>
</NFV>
```

Listato A.14. Risposta fornita dall'ASTRID Security Controller all'invio della richiesta di aggiunta istanza sull'endpoint *register/event*


```
[
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.147",
                "id": "insert",
                "n": "rule1@apache-25fhjb7w63-8rb1s",
                "src": "172.20.1.150",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@apache-25fhjb7w63-8rb1s",
                "src": "172.20.1.147",
                "timestamp": "2020-11-17T08:54:57"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "apache-25fhjb7w63-8rb1s",
        "id": "firewall@apache-25fhjb7w63-8rb1s",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule1@apache-58fc12sdd4-1b6v5",
                "src": "172.20.1.155",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.155",
                "id": "insert",
                "n": "rule2@apache-58fc12sdd4-1b6v5",
                "src": "172.20.1.150",
                "timestamp": "2020-11-17T08:54:57"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "apache-58fc12sdd4-1b6v5",
```

```
        "id": "firewall@apache-58fc12sdd4-1b6v5",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.0/24",
                "id": "insert",
                "n": "rule1@mysql-v4f8ss32ce-2f5v4",
                "src": "172.20.1.0/24",
                "timestamp": "2020-11-17T08:54:57"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "mysql-v4f8ss32ce-2f5v4",
        "id": "firewall@mysql-v4f8ss32ce-2f5v4",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.152",
                "id": "insert",
                "n": "rule1@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.150",
                "timestamp": "2020-11-17T08:54:57"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.152",
                "timestamp": "2020-11-17T08:54:57"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "nginx-ed46c14d85-rf12v",
        "id": "firewall@nginx-ed46c14d85-rf12v",
        "status": "started"
    }
]
```


Listato A.15. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutti gli *ASTRID Agents* dopo aver inviato un evento di aggiunta istanza

```
2020-11-17 08:53:52.247 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.servlet.
↪ DispatcherServlet        : POST "/controller/register/event", parameters={}
```

```
2020-11-17 08:53:52.776 DEBUG 14350 --- [http-nio-8083-exec-6] s.w.s.m.m.a.
↪ RequestMappingHandlerMapping : Mapped to public org.springframework.http.
↪ ResponseEntity<it.polito.verefoo.jaxb.NFV> it.polito.astrid.controllers.
↪ RegistrationController.registerEvent(it.polito.verefoo.astrid.jaxb.
↪ InfrastructureEvent) throws it.polito.astrid.controllers.
↪ ResourceNotFoundException,it.polito.astrid.controllers.
↪ AstridComponentNotFoundException
2020-11-17 08:53:52.864 DEBUG 14350 --- [http-nio-8083-exec-6] m.m.a.
↪ RequestResponseBodyMethodProcessor : Read "application/xml;charset=UTF-8" to [
↪ it.polito.verefoo.astrid.jaxb.InfrastructureEvent4d758610]
2020-11-17 08:53:52.867  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.c.
↪ RegistrationController           : There isn't any Astrid Database configured.
2020-11-17 08:53:52.976  INFO 14350 --- [http-nio-8083-exec-6] i.p.astrid.service
↪ .RegisterEventService  : ++++++++++ registerEvent Controller Obtained Event of
↪ type: add
2020-11-17 08:53:52.977  INFO 14350 --- [http-nio-8083-exec-6] i.p.astrid.service
↪ .RegisterEventService  : ++++++++++ There isn't any Astrid DB configured!
2020-11-17 08:53:52.977  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ ask to Context Broker the infrastructure
↪  info of graph: test-network
2020-11-17 08:53:53.092 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/connection
2020-11-17 08:53:53.093 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
2020-11-17 08:53:53.125 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:53:53.127 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-17 08:53:53.129  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : ++++++++++ Context Broker answer successfully with
↪  connection of requested network
2020-11-17 08:53:53.140 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/exec-env/apache
↪ -25fhjb7w63-8rb1s
2020-11-17 08:53:53.140 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
2020-11-17 08:53:53.158 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:53:53.160 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-17 08:53:53.167 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/exec-env/mysql-
↪ v4f8ss32ce-2f5v4
2020-11-17 08:53:53.168 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
2020-11-17 08:53:53.187 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:53:53.188 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-17 08:53:53.189 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/exec-env/nginx-
↪ ed46c14d85-rf12v
2020-11-17 08:53:53.190 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
2020-11-17 08:53:53.211 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:53:53.212 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-17 08:53:53.214 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/exec-env/apache
↪ -58fc12sdd4-1b6v5
2020-11-17 08:53:53.214 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
```

```
2020-11-17 08:53:53.233 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:53:53.234 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-17 08:53:53.235  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.s.
↪ RegisterInfrastructureService    : ++++++++++ registerInfrastructure Controller
↪  Obtained Infrastructure info for: test-network
```

Listato A.16. File di log dell'ASTRID Security Controller dopo aver inviato un evento di aggiunta istanza - Prima parte

```
2020-11-17 08:53:53.258  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.s.
↪ RegisterInfrastructureService    : ++++++++++ registerInfrastructure Controller
↪  Sending Infrastructure info to Verefoo
2020-11-17 08:53:53.259 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP POST http://localhost:8085/verefoo/dc
2020-11-17 08:53:53.259 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[*/*]
2020-11-17 08:53:53.260 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Writing [it.polito.verefoo.astrid.jaxb.
↪ InfrastructureInfo6c924e17] as "application/xml"
2020-11-17 08:54:57.391 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:54:57.404 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [it.polito.verefoo.jaxb.NFV] as "
↪ application/xml"
2020-11-17 08:54:57.691  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ started to instance the firewalls
2020-11-17 08:54:57.712 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP GET http://130.251.17.128:5000/instance/agent
2020-11-17 08:54:57.715 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Accept=[text/plain, */*]
2020-11-17 08:54:57.750 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:54:57.751 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
2020-11-17 08:54:57.763  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID:
↪ apache-25fhjb7w63-8rb1s already exist
2020-11-17 08:54:57.764  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: apache-25
↪ fhjb7w63-8rb1s already exist. Updating in progress...
2020-11-17 08:54:57.767 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : HTTP PUT http://130.251.17.128:5000/instance/agent
2020-11-17 08:54:57.768 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Writing [{"id":"firewall@apache-25fhjb7w63-8rb1s","
↪ actions":[{"action":"DENY","id":"default","timestamp":"2020-11-17T08:54:57"},{"
↪ dst":"172.20.1.147","src":"172.20.1.150","action":"ALLOW","id":"insert","n":"
↪ rule1@apache-25fhjb7w63-8rb1s","timestamp":"2020-11-17T08:54:57"},{"dst":"
↪ 172.20.1.150","src":"172.20.1.147","action":"ALLOW","id":"insert","n":"
↪ rule2@apache-25fhjb7w63-8rb1s","timestamp":"2020-11-17T08:54:57"}]}] as "
↪ application/json"
2020-11-17 08:54:57.857 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
↪ RestTemplate               : Response 200 OK
2020-11-17 08:54:57.858  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully updated!
2020-11-17 08:54:57.859  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID:
↪ apache-58fc12sdd4-1b6v5 already exist
2020-11-17 08:54:57.859  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: apache-58
↪ fc12sdd4-1b6v5 not exist. Creating in progress...
```

```
56 2020-11-17 08:54:57.860 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : HTTP POST http://130.251.17.128:5000/instance/agent
57 2020-11-17 08:54:57.861 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Accept=[text/plain, */*]
58 2020-11-17 08:54:57.863 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Writing [{"exec_env_id":"apache-58fc12sdd4-1b6v5","
   ↪ agent_catalog_id":"firewall","id":"firewall@apache-58fc12sdd4-1b6v5","actions":
   ↪ [{"action":"DENY","id":"default","timestamp":"2020-11-17T08:54:57"},{"dst":"
   ↪ 172.20.1.150","src":"172.20.1.155","action":"ALLOW","id":"insert","n":"
   ↪ rule1@apache-58fc12sdd4-1b6v5","timestamp":"2020-11-17T08:54:57"},{"dst":"
   ↪ 172.20.1.155","src":"172.20.1.150","action":"ALLOW","id":"insert","n":"
   ↪ rule2@apache-58fc12sdd4-1b6v5","timestamp":"2020-11-17T08:54:57"}],"status":"
   ↪ started"}] as "application/json"
59 2020-11-17 08:54:58.467 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Response 201 CREATED
60 2020-11-17 08:54:58.468 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Reading to [java.lang.String] as "application/json"
61 2020-11-17 08:54:58.469  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully instanced!
62 2020-11-17 08:54:58.469  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID: mysql
   ↪ -v4f8ss32ce-2f5v4 already exist
63 2020-11-17 08:54:58.469  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: mysql-
   ↪ v4f8ss32ce-2f5v4 already exist. Updating in progress...
64 2020-11-17 08:54:58.472 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : HTTP PUT http://130.251.17.128:5000/instance/agent
65 2020-11-17 08:54:58.473 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Writing [{"id":"firewall@mysql-v4f8ss32ce-2f5v4","
   ↪ actions":[{"action":"DENY","id":"default","timestamp":"2020-11-17T08:54:57"},{"
   ↪ dst":"172.20.1.0\/24","src":"172.20.1.0\/24","action":"ALLOW","id":"insert","n"
   ↪ :"rule1@mysql-v4f8ss32ce-2f5v4","timestamp":"2020-11-17T08:54:57"}]}] as "
   ↪ application/json"
66 2020-11-17 08:54:58.512 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Response 200 OK
67 2020-11-17 08:54:58.513  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully updated!
68 2020-11-17 08:54:58.513  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ check if firewall of node with ID: nginx
   ↪ -ed46c14d85-rf12v already exist
69 2020-11-17 08:54:58.514  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ firewall of node with ID: nginx-
   ↪ ed46c14d85-rf12v already exist. Updating in progress...
70 2020-11-17 08:54:58.516 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : HTTP PUT http://130.251.17.128:5000/instance/agent
71 2020-11-17 08:54:58.517 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Writing [{"id":"firewall@nginx-ed46c14d85-rf12v","
   ↪ actions":[{"action":"DENY","id":"default","timestamp":"2020-11-17T08:54:57"},{"
   ↪ dst":"172.20.1.152","src":"172.20.1.150","action":"ALLOW","id":"insert","n":"
   ↪ rule1@nginx-ed46c14d85-rf12v","timestamp":"2020-11-17T08:54:57"},{"dst":"
   ↪ 172.20.1.150","src":"172.20.1.152","action":"ALLOW","id":"insert","n":"
   ↪ rule2@nginx-ed46c14d85-rf12v","timestamp":"2020-11-17T08:54:57"}]}] as "
   ↪ application/json"
72 2020-11-17 08:54:58.563 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.client.
   ↪ RestTemplate               : Response 200 OK
73 2020-11-17 08:54:58.564  INFO 14350 --- [http-nio-8083-exec-6] i.p.a.service.
   ↪ InfrastructureInfoRequest  : +++++++++ Firewall successfully updated!
74 2020-11-17 08:54:58.569 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.w.s.m.m.a.
   ↪ HttpEntityMethodProcessor  : Found 'Content-Type:application/xml' in response
75 2020-11-17 08:54:58.570 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.w.s.m.m.a.
   ↪ HttpEntityMethodProcessor  : Writing [it.polito.verefoo.jaxb.NFV1896a886]
76 2020-11-17 08:54:58.584 DEBUG 14350 --- [http-nio-8083-exec-6] o.s.web.servlet.
   ↪ DispatcherServlet          : Completed 200 OK
```

Listato A.17. File di log dell'ASTRID Security Controller dopo aver inviato un evento di aggiunta istanza - Seconda parte

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NFV>
    <graphs>
        <graph id="29">
            <node name="172.20.1.150" functional_type="WEBSERVER">
                <neighbour name="172.20.1.170"/>
                <configuration name="mysql" description="8000">
                    <webserver>
                        <name>172.20.1.150</name>
                    </webserver>
                </configuration>
            </node>
            <node name="172.20.1.170" functional_type="FIREWALL">
                <neighbour name="172.20.1.150"/>
                <neighbour name="172.20.1.172"/>
                <neighbour name="172.20.1.175"/>
                <configuration name="mysql-v4f8ss32ce-2f5v4" description="
↪ 172.20.1.150">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>172.20.1.-1</source>
                            <destination>172.20.1.-1</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                    </firewall>
                </configuration>
            </node>
            <node name="172.20.1.152" functional_type="WEBSERVER">
                <neighbour name="172.20.1.172"/>
                <configuration name="nginx" description="8080">
                    <webserver>
                        <name>172.20.1.152</name>
                    </webserver>
                </configuration>
            </node>
            <node name="172.20.1.172" functional_type="FIREWALL">
                <neighbour name="172.20.1.152"/>
                <neighbour name="172.20.1.170"/>
                <neighbour name="172.20.1.175"/>
                <configuration name="nginx-ed46c14d85-rf12v" description="
↪ 172.20.1.152">
                    <firewall defaultAction="DENY">
                        <elements>
                            <action>ALLOW</action>
                            <source>172.20.1.150</source>
                            <destination>172.20.1.152</destination>
                            <protocol>ANY</protocol>
                            <src_port>*</src_port>
                            <dst_port>*</dst_port>
                        </elements>
                        <elements>
                            <action>ALLOW</action>
                            <source>172.20.1.152</source>
                            <destination>172.20.1.150</destination>
                            <protocol>ANY</protocol>
```

```
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
        <node name="172.20.1.155" functional_type="WEBSERVER">
            <neighbour name="172.20.1.175"/>
            <configuration name="apache" description="80">
                <webserver>
                    <name>172.20.1.155</name>
                </webserver>
            </configuration>
        </node>
        <node name="172.20.1.175" functional_type="FIREWALL">
            <neighbour name="172.20.1.155"/>
            <neighbour name="172.20.1.170"/>
            <neighbour name="172.20.1.172"/>
            <configuration name="apache-58fc12sdd4-1b6v5" description="
↪ 172.20.1.155">
                <firewall defaultAction="DENY">
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.155</source>
                        <destination>172.20.1.150</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                    <elements>
                        <action>ALLOW</action>
                        <source>172.20.1.150</source>
                        <destination>172.20.1.155</destination>
                        <protocol>ANY</protocol>
                        <src_port>*</src_port>
                        <dst_port>*</dst_port>
                    </elements>
                </firewall>
            </configuration>
        </node>
    </graph>
</graphs>
<Constraints>
    <NodeConstraints/>
    <LinkConstraints/>
</Constraints>
<PropertyDefinition>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.155" dst="
↪ 172.20.1.150" src_port="80" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.155" src_port="8000" dst_port="80" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.152" dst="
↪ 172.20.1.150" src_port="8080" dst_port="8000" isSat="true"/>
    <Property name="ReachabilityProperty" graph="29" src="172.20.1.150" dst="
↪ 172.20.1.152" src_port="8000" dst_port="8080" isSat="true"/>
</PropertyDefinition>
</NFV>
```

Listato A.18. Risposta fornita dall'ASTRID Security Controller all'invio della richiesta di eliminazione istanza sull'endpoint *register/event*

```
1 [
```

```
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T02:14:02"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.0/24",
                "id": "insert",
                "n": "rule1@mysql-v4f8ss32ce-2f5v4",
                "src": "172.20.1.0/24",
                "timestamp": "2020-11-17T02:14:02"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "mysql-v4f8ss32ce-2f5v4",
        "id": "firewall@mysql-v4f8ss32ce-2f5v4",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T02:14:02"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.152",
                "id": "insert",
                "n": "rule1@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.150",
                "timestamp": "2020-11-17T02:14:02"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.152",
                "timestamp": "2020-11-17T02:14:02"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "nginx-ed46c14d85-rf12v",
        "id": "firewall@nginx-ed46c14d85-rf12v",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-11-17T02:14:02"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule1@apache-58fc12sdd4-1b6v5",
                "src": "172.20.1.155",
```

```
                "timestamp": "2020-11-17T02:14:02"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.155",
                "id": "insert",
                "n": "rule2@apache-58fc12sdd4-1b6v5",
                "src": "172.20.1.150",
                "timestamp": "2020-11-17T02:14:02"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "apache-58fc12sdd4-1b6v5",
        "id": "firewall@apache-58fc12sdd4-1b6v5",
        "status": "started"
    }
]
```

Listato A.19. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutti gli *ASTRID Agents* dopo aver inviato un evento di rimozione istanza

```
public void instanceFirewallFromKafkaEvent(KafkaMessage kmes) throws ContextBrokerException{
    logger.info("+++++++++ DoS LOIC attack detected!");
    List<Agent_Instance> agentInstance = new ArrayList<>();
    List<Execution_Environment> exec_env = new ArrayList<>();
    int i = 0;
    Date dateNow = new Date();
    SimpleDateFormat ft = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss");
    RestTemplate restTemplate = new RestTemplate();
    Jaxb2RootElementHttpMessageConverter converter = new Jaxb2RootElementHttpMessageConverter();
    converter.setSupportedMediaTypes(Collections.singletonList(MediaType.ALL));
    restTemplate.setMessageConverters(Arrays.asList(converter, new StringHttpMessageConverter()));
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> requestBody = null;
    ResponseEntity<String> result = null;
    JSONObject query = null;
    JSONObject defAction = null;
    JSONObject rule = null;
    JSONArray actions = null;
    List<JSONObject> rules = new ArrayList<>();
    boolean trovato = false;
    //recovery the agent instance
    agentInstance = getAgentInstance();
    //recovery the execution environment
    exec_env = getExecutionEnvironment();
    Iterator<Execution_Environment> it = exec_env.iterator();

    //recovery the id of exec_env that has the IP egual to source IP of the kafka message
    while(it.hasNext() && trovato==false) {
        Execution_Environment node = it.next();
        if((node.getHostname().compareTo(kmes.getSource_ip())==0) || (node.getHostname().compareTo(kmes.getDestination_ip())==0)) {
            trovato = true;
            query = new JSONObject();
            defAction = new JSONObject();
            actions = new JSONArray();
```

```
36              //check if already exist a firewall of that node
37              if (checkFirewallExist(node.getId(), agentInstance)==true) {
38                  //firewall already exist, so update the rules
39                  Agent_Instance fw = getFirewallwithExecEnvId(node.getId(),
↪ agentInstance);
40                  try {
41                      query.put("id", "firewall@" + node.getId());
42                      //retrieve the action list
43                      List<actions> act = fw.getActions();
44                      Iterator<actions> act_it = act.iterator();
45                      i=0;
46                      while(act_it.hasNext()) {
47                          actions action = act_it.next();
48                          rule = new JSONObject();
49                          try {
50                              rule.put("id", action.getId());
51                              if (action.getN() != null)
52                                  rule.put("n", action.getN());
53                              if(action.getSrc() != null)
54                                  rule.put("src", action.getSrc());
55                              if(action.getDst() != null)
56                                  rule.put("dst", action.getDst());
57                              if(action.getAction() != null)
58                                  rule.put("action", action.getAction());
59                              if(action.getTimestamp() != null)
60                                  rule.put("timestamp", action.getTimestamp());
61                              rules.add(rule);
62                              actions.put(rules.get(i));
63                              i++;
64                          }catch(Exception ex) {
65                              logger.error("+++++++++ error while costruct the
↪ query: " + ex.getMessage());
66                              throw new ContextBrokerException(ex.getMessage());
67                          }
68                      }
69                      rule = new JSONObject();
70                      if(node.getHostname().compareTo(kmes.getSource_ip())==0) {
71                          //the node is the sender of the attack
72                          rule.put("id", "insert");
73                          rule.put("n", "rule" + i + "@" + node.getId());
74                          rule.put("src", node.getHostname());
75                          rule.put("dst", kmes.getDestination_port());
76                          rule.put("action", "DENY");
77                          rule.put("timestamp", ft.format(dateNow));
78                      }else {
79                          //the node is the target of the attack
80                          rule.put("id", "insert");
81                          rule.put("n", "rule" + i + "@" + node.getId());
82                          rule.put("src", kmes.getSource_ip());
83                          rule.put("dst", node.getHostname());
84                          rule.put("action", "DENY");
85                          rule.put("timestamp", ft.format(dateNow));
86                      }
87                      rules.add(rule);
88                      actions.put(rules.get(i));
89                      i++;
90                      query.put("actions", actions);
91                  }catch(Exception ex) {
92                      logger.error("+++++++++ error while costruct query: " + ex.
↪ getMessage());
93                      throw new ContextBrokerException(ex.getMessage());
94                  }
95                  //send the request
```

```
96                  requestBody = new HttpEntity<String>(query.toString(), headers);
97                  try {
98                      restTemplate.put("http://" + ContextBroker.getIPAddress() + "
↪ :" + ContextBroker.getPort() + "/instance/agent", requestBody);
99                      logger.info("+++++++++ Firewall to block attack successfully
↪ updated!");
100                 }catch (HttpClientErrorException e) {
101                     if(e.getStatusCode() == HttpStatus.UNPROCESSABLE_ENTITY) {
102                         logger.info("+++++++++ Unable to reach the firewall, but
↪ insert it into Context Broker");
103                     }else {
104                         logger.error("+++++++++ error while instance the
↪ firewalls to block attack: " + e.getMessage());
105                         throw new ContextBrokerException(e.getMessage());
106                     }
107                 }catch (HttpServerErrorException e) {
108                     logger.error("+++++++++ error while instance the firewalls to
↪  block attack: " + e.getMessage());
109                     throw new ContextBrokerException(e.getMessage());
110                 }
111
112             }else {
113                 //firewall doesn't exist, so create it
114                 try {
115                     query.put("id", "firewall@" + node.getId());
116                     query.put("agent_catalog_id", "firewall");
117                     query.put("exec_env_id", node.getId());
118                     query.put("status", "started");
119                     defAction.put("id", "default");
120                     defAction.put("action", "DENY");
121                     defAction.put("timestamp", ft.format(dateNow));
122                     actions.put(defAction);
123                     query.put("actions", actions);
124                 }catch(Exception ex) {
125                     logger.error("+++++++++ error while costruct the query: " +
↪ ex.getMessage());
126                     throw new ContextBrokerException(ex.getMessage());
127                 }
128                 //send the request
129                 requestBody = new HttpEntity<String>(query.toString(), headers);
130                 try {
131                     result = restTemplate.postForEntity("http://" + ContextBroker
↪ .getIPAddress() + ":" + ContextBroker.getPort() + "/instance/agent",
↪ requestBody, String.class);
132                     logger.info("+++++++++ Firewall to block attack successfully
↪ instanced!");
133                 }catch (HttpClientErrorException e) {
134                     if(e.getStatusCode() == HttpStatus.UNPROCESSABLE_ENTITY) {
135                         logger.info("+++++++++ Unable to reach the firewall, but
↪ insert it into Context Broker");
136                     }else {
137                         logger.error("+++++++++ error while instance the firewall
↪  to block attack: " + e.getMessage());
138                         throw new ContextBrokerException(e.getMessage());
139                     }
140                 }catch (HttpServerErrorException e) {
141                     logger.error("+++++++++ error while instance the firewall to
↪ block attack: " + e.getMessage());
142                     throw new ContextBrokerException(e.getMessage());
143                 }
144             }
145         }
146     }
```

```
147 }
```

Listato A.20. Codice del metodo instanceFirewallFromKafkaEvent


```
[
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-12-03T02:14:42"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.152",
                "id": "insert",
                "n": "rule1@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.150",
                "timestamp": "2020-12-03T02:14:42"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@nginx-ed46c14d85-rf12v",
                "src": "172.20.1.152",
                "timestamp": "2020-12-03T02:14:42"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "nginx-ed46c14d85-rf12v",
        "id": "firewall@nginx-ed46c14d85-rf12v",
        "status": "started"
    },
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-12-03T02:14:42"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.147",
                "id": "insert",
                "n": "rule1@apache-25fhjb7w63-8rb1s",
                "src": "172.20.1.150",
                "timestamp": "2020-12-03T02:14:42"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@apache-25fhjb7w63-8rb1s",
                "src": "172.20.1.147",
                "timestamp": "2020-12-03T02:14:42"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "apache-25fhjb7w63-8rb1s",
        "id": "firewall@apache-25fhjb7w63-8rb1s",
        "status": "started"
    },
```

```
    {
        "actions": [
            {
                "action": "DENY",
                "id": "default",
                "timestamp": "2020-12-03T02:14:42"
            },
            {
                "action": "ALLOW",
                "dst": "172.20.1.0/24",
                "id": "insert",
                "n": "rule1@mysql-v4f8ss32ce-2f5v4",
                "src": "172.20.1.0/24",
                "timestamp": "2020-12-03T02:14:42"
            },
            {
                "action": "DENY",
                "dst": "172.20.1.150",
                "id": "insert",
                "n": "rule2@mysql-v4f8ss32ce-2f5v4",
                "src": "172.20.1.5",
                "timestamp": "2020-12-03T02:28:23"
            }
        ],
        "agent_catalog_id": "firewall",
        "exec_env_id": "mysql-v4f8ss32ce-2f5v4",
        "id": "firewall@mysql-v4f8ss32ce-2f5v4",
        "status": "started"
    }
]
```


Listato A.21. Risposta fornita dall'ASTRID Context Broker quando viene richiesto l'elenco di tutti gli *ASTRID Agents* dopo che il Security Controller ha ricevuto una notifica di attacco di tipo DoS