Monitoraggio di rete con tecnologie SDN (Software-Defined Networking)

Aprile 2021

Autore: Giacomo Ondesca (POLITO)

Supervisore: Gabriele Castellano, Fulvio Risso (POLITO)

Coordinatori del progetto: Fabrizio Garrone (RSE), Fulvio Risso (POLITO)

Indice dei contenuti

1 Abstract

In the networking world, the capability to easily monitor, manage, and program many network devices is gaining attention. The paradigm called Software Defined Network (SDN) is the emerging architecture that takes over this challenge. It allows to separate the network intelligence control logic from the physical data-plane appliance, providing a centralized point of control, called SDN controller, that is able to monitor the network behavior and dynamically react to changes with new network configurations. So, the underlying hardware technologies are still there, but they are programmed by means of third-party software applications. In fact, the power of the SDN technology relies on internal services that expose APIs to applications to help them manage endpoints and the entire infrastructure with higher speed and flexibility.

The aim of this document is to deploy the SDN paradigm in a "traditional" energy distribution infrastructure, to control the ICT network and react to different events (e.g., fault or security attacks) with the proper counter-measure (e.g., re-route the traffic or isolate the malicious source).

This document will start by presenting the main characteristics of some application/network data monitoring technologies. The SDN controller needs to establish a connection not only with traditional network devices such as routers or switches, but also with other ICT electrical components (e.g., embedded devices used in the electrical grid), which are equipped with software probes and monitoring tools in order to complete the information about their network/application activities and internal working status. All the generated monitoring data is collected in a unique datastore that is also responsible to analyze them and give the proper information to the controller. Furthermore, this document will analyze the available SDN controllers, with a special focus on the ones with an open architecture that are more suitable for research and experimentation purposes, presenting the different offered features and their overall architecture.

Finally, this document will propose an overall architecture that can be used to control the ICT network infrastructure, with the objectives mentioned above. The operability and responsiveness of the proposed solution will be achieved by the definition and implementation of two use cases that trigger the control logic into action. They enclose aspects ranging from security, isolating the node that issues malicious traffic, to redundancy and fault tolerance, redirecting the stream of data toward "backup" services in case of catastrophic events.

2 Introduzione

La finalità di questa attività di ricerca prevede l'individuazione e la convalidazione di una soluzione software collocata in una rete elettrica intelligente in grado di automatizzare il controllo e la gestione sia di classici dispositivi di rete, quali router e switch, che apparati tipici di una SG (Smart Grid) quali PMU, IED, ecc.. In particolare, questa tecnologia deve consentire di monitorare lo stato e il pattern di traffico generato dai dispositivi incaricati del monitoraggio della rete elettrica, andando ad installare opportune sonde software su di essi con la capacità di comunicare tali informazioni ad un'unità logica in grado di analizzarle e interpretarle, per poi attuare una retroazione sulla rete per modificarne il comportamento ove necessario, mitigando l'impatto di eventuali attacchi di sicurezza o guasti hardware. Questo permetterebbe non solo di individuare preventivamente eventuali malfunzionamenti hardware/software che consentirebbero un intervento tempestivo di sostituzione/redeploy dell'apparato e rerouting automatico del traffico, ma scoprire eventuali comportamenti anomali da parte dei dispositivi, isolando l'oggetto malevolo dal resto della rete. Questa tecnologia può svolgere quindi un ruolo determinante nel garantire una maggior sicurezza dell'intero sistema. A questo proposito sono stati definiti nel documento due possibili eventi, i più usuali e rilevanti nelle Smart Grid, atti a testare l'operatività della logica automatica di controllo interrogando ed attingendo ai dati di monitoraggio immagazzinati (Sezione 2). Il primo evento ha lo scopo di verificare la capacità della soluzione software sviluppata di riuscire a rintracciare eventuali attacchi informatici con l'intenzione di alterare il comportamento della rete, di rallentare le performance o addirittura paralizzare l'intero funzionamento, rubare dati sensibili o qualsiasi altro obiettivo non in linea con le policy del sistema. Il secondo evento, di natura non intenzionale, riguarda il livello di dinamicità raggiunto dall'infrastruttura al far fronte a guasti software o hardware, proponendo percorsi alternativi e componenti ausiliari nell'ipotesi in cui gli elementi principali non risultino operativi.

In questo documento verranno prima analizzati i possibili componenti tecnologici per realizzare il sistema descritto in precedenza, per poi definire una visione architetturale completa di una possibile soluzione al problema in esame, evidenziando i componenti principali e le interazioni tra di essi. Infine, verrà presentato lo una prima proposta relativa allo sviluppo dell'unità di controllo che, analizzando i dati raccolti, inietta delle regole ("*policies*") in grado di influenzare il funzionamento della rete.

3 Stato dell'arte

In questo capitolo vengono trattate, dal punto di vista teorico, le principali tecnologie che si prestano maggiormente ad essere utilizzate nello scenario presentato.

Nella prima parte si rivolge l'attenzione ai controller SDN (Software Defined Networking), svolgendo un lavoro di selezione tra quelli disponibili nel mondo open-source.

Successivamente, si illustrerà il risultato di una ricerca bibliografica atta a ricercare le attuali soluzioni disponibili ed adottate in produzione per quanto riguarda il monitoraggio di sistema, distinguendo le tecnologie per il monitoraggio di informazioni a livello applicativo (di processo software) da quelle per il monitoraggio di informazioni a livello di rete, mostrando infine dove ricade l'impiego di ognuna di esse sui dispositivi presenti nella rete elettrica.

Controller SDN e tool di monitoraggio risultano elementi complementari per finalizzare l'automatizzazione della gestione della rete. Sebbene l'oggetto principale di studio su cui sono rivolti gli sforzi e le attenzioni di questo lavoro sia lo sviluppo di una logica di controllo centralizzata, questa se non avesse a disposizione le informazioni di stato dei dispositivi e del traffico generato non saprebbe quando, dove e come intervenire.

3.1 SDN

SDN è un paradigma utilizzato per configurare e gestire il comportamento delle reti, attraverso un controller software centralizzato. In prima battuta si presentano quindi i concetti generali, successivamente le iniziative guidate dal mondo open source per l'implementazione di controller ed infine lo stato attuale in letteratura riguardo l'applicazione di tale tecnologia per gestire la comunicazione tra le entità in ambiente SG.

La decisione di utilizzare una delle piattaforme open source ha degli ovvi motivi. Nella scelta tra opzioni proprietarie ed open source non c'è una risposta giusta o sbagliata, il tutto si riduce a cosa l'azienda/organizzazione necessita di raggiungere. Le soluzioni commerciali si sono adattate al mondo open, ed esportano API che permettono l'integrazione del loro prodotto con componenti di rete non proprietari, mantenendo però controllo sul codice sorgente. Questo implica che è il fornitore del controller che detta quali features esso offre e quando ne vengono rilasciate di nuove. D'altro canto, si ha il supporto tecnico nel caso si dovessero verificare malfunzionamenti al dispositivo. Essendo questa però un'attività di ricerca e sperimentazione, può essere necessario poter metter mano al codice sorgente del controller per implementare una soluzione custom più adatta al nostro ambiente, pertanto il software SDN open source permette un'evoluzione flessibile che segue i propri ritmi di sviluppo del codice. Infine, è da notare come solitamente le modifiche al software open-source che si rivelano interessanti vengono spesso riprese dalle soluzioni proprietarie, incorporandole nei vari prodotti in una successiva fase temporale. Questo permette alle aziende "utilizzatori" (es. gestori elettrici) di poter disporre di tali tecnologie nel momento in cui introdurranno questi concetti nelle loro infrastrutture dati di produzione.

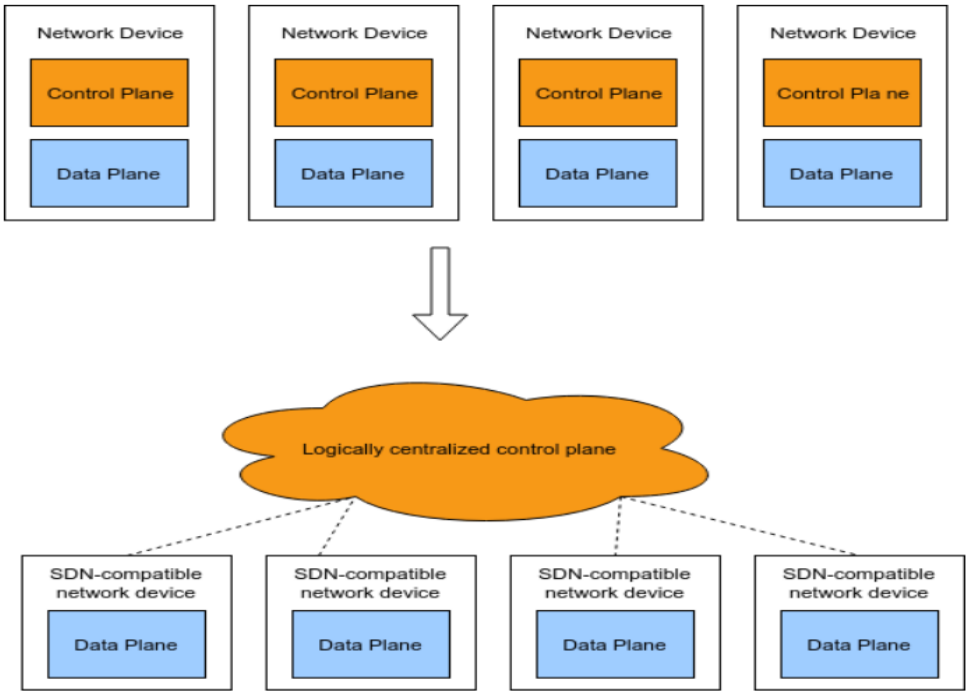
3.1.1 Concetti generali

SDN (Software Defined Networking) è un paradigma software che permette agli amministratori di rete di gestire e controllare in modo dinamico e automatico un elevato numero di dispositivi di rete, i servizi, la topologia, il percorso del traffico, le policy di gestione dei pacchetti (QoS) utilizzando APIs spesso di tipo "intent-based" (ossia funzionali anziché dichiarative), linguaggi di alto livello e software di terze parti.

L'idea principale inizialmente in SDN consisteva nella separazione fisica del control plane dal data plane dalle appliance fisiche di rete:

- Data plane (switching fabric) - unità predisposta all'invio dei pacchetti nella rete;
- Control plane (control unit) - unità che decide come il forwarding deve essere eseguito.

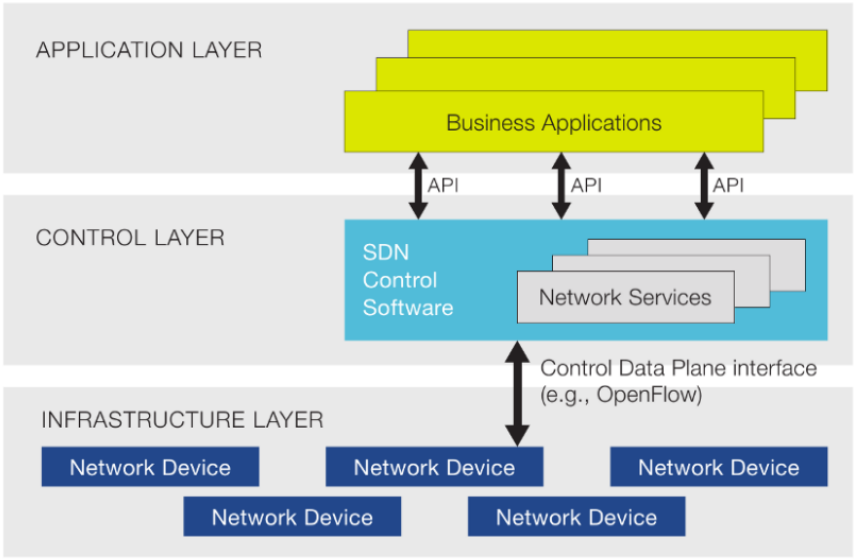
La funzione di controllo viene eliminata dal dispositivo e messa in un'unità logica centralizzata chiamata SDN controller.



La separazione del data plane dal control plane ha preso piede in pochi ambienti (quindi i dispositivi attuali mantengono ancora localmente il control plane) ma è presente un'unità logica centralizzata (l'SDN controller) che mantiene una visibilità globale della rete, nasconde i dettagli dei diversi dispositivi di rete e permette di definire delle policy per sovrascrivere le decisioni prese localmente dal dispositivo.

L'architettura SDN separa la rete in tre differenti sezioni:

- **SDN controller** è il cervello della rete e attua da mediatore tra le applicazioni e le risorse di rete. Esso contiene un insieme di moduli che possono eseguire diversi tasks (es. quali dispositivi sono presenti nella rete, informazioni sulla topologia, ottenere informazioni statistiche su di essi, ecc…).
- **Southbound APIs.** Definisce un modo standard per il controller per interagire con i dispositivi; tuttavia non è mai stato definita un'interfaccia fisica precisa (lo stesso protocollo OpenFlow è scarsamente utilizzato) e pertanto il numero di protocolli utilizzato per interagire con i dispositivi finali (es. apparati di rete, host) è ampio e variegato. A causa di questa varietà, anche le azioni che possono essere fatte sui dispositivi di rete sono diverse, che dipendono dal protocollo stesso. Ad esempio, nel caso di OpenFlow il controller installa sui dispositivi delle regole di forwarding, nel caso di NetCONF è potenzialmente possibile configurare servizi generici, e così via.
- **Northboud APIs.** Il controller fornisce delle API che le applicazioni esterne, di più alto livello, utilizzano per programmare la rete. A sua volta, il controller traduce queste richieste in istruzioni a basso livello verso i dispositivi stessi**.** In questo modo le applicazioni non devono preoccuparsi degli aspetti a basso livello di ogni nodo, basta che comunichino con il controller attraverso northbound API, che molto spesso non sono altro che delle primitive REST, secondo lo standard HTTP.



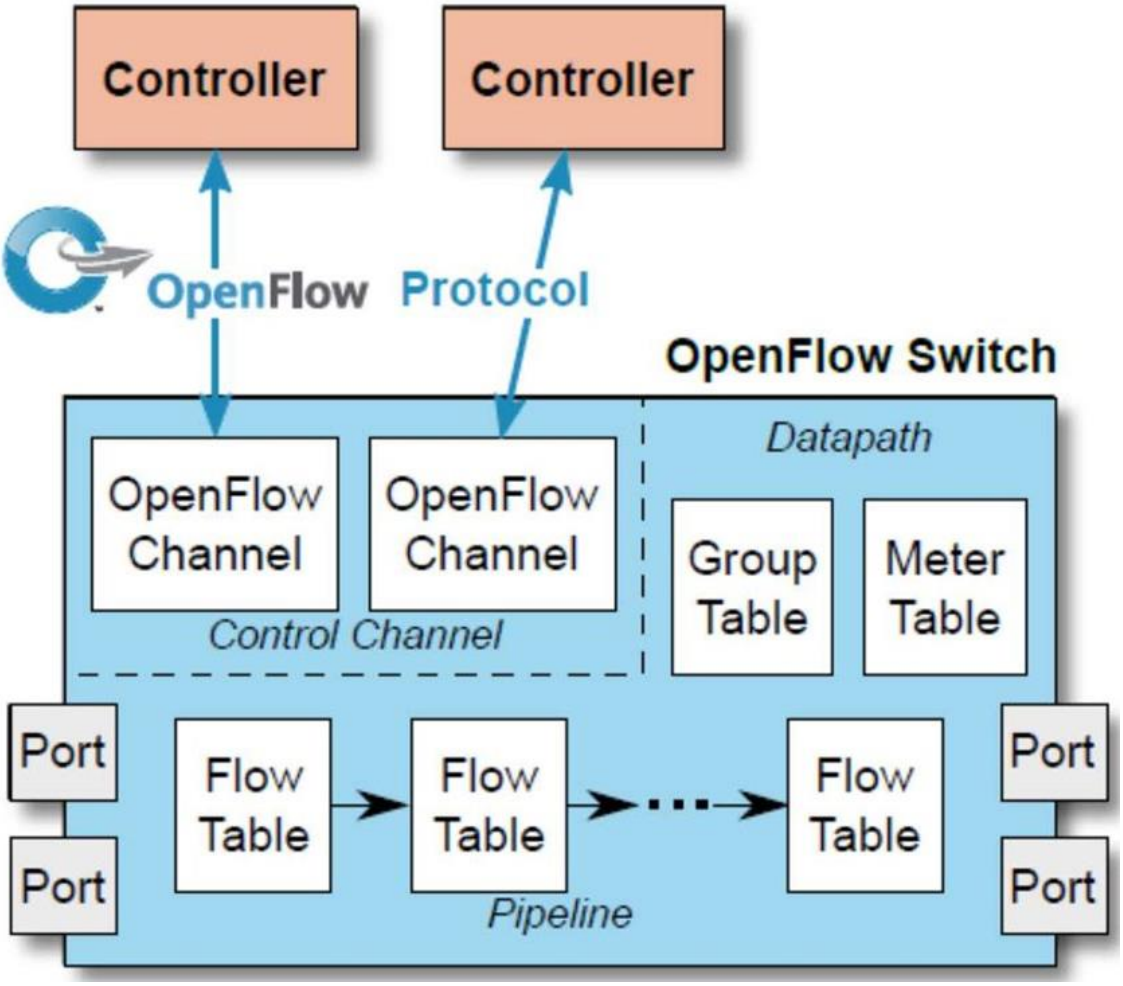
Il beneficio di SDN consiste nel fatto che si possono cambiare le regole di configurazione degli apparati (es. regole di forwarding nei router/switch OpenFlow) quando necessario (ad esempio per ridistribuire il carico del traffico in modo efficiente e flessibile), ridurre/aumentare la priorità del traffico o bloccare alcuni tipi di pacchetti. Inoltre, il controller fornisce una visibilità end-to-end (o a "big switch") della rete in modo tale che un amministratore di rete deve interagire con una sola entità, il controller, invece di configurare ogni singolo dispositivo.[1]

3.1.2 OpenFlow

Questa sezione descrive i requisiti che deve avere uno switch OpenFlow. Uno switch OF consiste in una o più flow table per eseguire il lookup dei pacchetti e il forwarding, e di un canale che impiega per la comunicazione verso l'SDN controller utilizzando appunto il protocollo OF. Con il protocollo OF, il controller può aggiungere, modificare ed eliminare le flow entry nelle flow table in modo reattivo (quindi in risposta ad un pacchetto) o in modo proattivo.

Ogni flow table nello switch contiene un insieme di flow entry, ognuna delle quali consiste nei campi match, counter, timeout per la sua durata e di istruzioni da applicare ai pacchetti che hanno effettuato il match. Il matching dei pacchetti inizia dalla prima flow table e può continuare con flow table addizionali per creare delle pipeline di processamento. Il processo di pipeline termina quando l'istruzione associata alla flow entry che ha effettuato il match non specifica una successiva flow table; a questo punto il pacchetto viene inoltrato o modificato.

[1] https://searchnetworking.techtarget.com/definition/software-defined-networking-SDN



Components of an OpenFlow switch

Le flow entry effettuano il match del pacchetto in ordine di priorità e si sceglie la flow entry che ha priorità più alta. Se un match viene trovato, la corrispondenti istruzioni associate alla flow entry vengono eseguite ed il contatore associato viene aggiornato. Al contrario, se non c'è nessun match nella flow table, il pacchetto viene gestito in base alla configurazione della flow entry denominata table-miss: per esempio, il pacchetto può essere trasmesso al controller attraverso il canale OF oppure può essere scartato. Essa ha la priorità più bassa (zero) ed non effettua corrispondenze su nessun campo.

I campi dell'header del pacchetto utilizzati per il match della flow entry dipendono dal tipo del pacchetto, e tipicamente includono vari campi degli header dei classici protocolli di rete quali l'indirizzo sorgente Ethernet o l'indirizzo destinazione IPv4. In aggiunta la corrispondenza può avvenire attraverso la porta di ingresso o metadati (utilizzati per scambiare informazioni tra le tabelle degli switch). Un pacchetto effettua un match con un flow entry se tutti i campi del match della flow entry hanno una corrispondenza con i valori dei campi dell'header del pacchetto.

Ogni istruzione contiene una serie di azioni per effettuare l'inoltro dei pacchetti o per modificarli e può anche cambiare la pipeline di processamento (cioè dirigere il pacchetto ad un altra flow table). Le istruzioni più note sono:

- Apply Action(s): applica una o più azioni specificate immediatamente. Questa istruzioni può essere utilizzata per per modificare il pacchetto tra due flow table o per eseguire più azioni dello stesso tipo. Le azioni sono specificate come un action list. Le azioni specificate nella lista sono eseguite in ordine, l'effetto è cumulativo e se la lista contiene un'azione di output, una copia del pacchetto nel suo stato corrente viene inoltrata alla porta desiderata. Dopo l'esecuzione della action list, l'esecuzione della pipeline di processamento continua sul pacchetto modificato.
- Goto-Table next-table-id: indica la prossima flow table nella pipeline di processamento. L'id della tabella deve essere più grande dell'id della tabella corrente. Ovviamente l'ultima la flow entry dell'ultima flow table non può includere questa istruzione.

L'insieme delle istruzioni associate ad una flow entry contiene al massimo un'istruzione per ogni tipo.

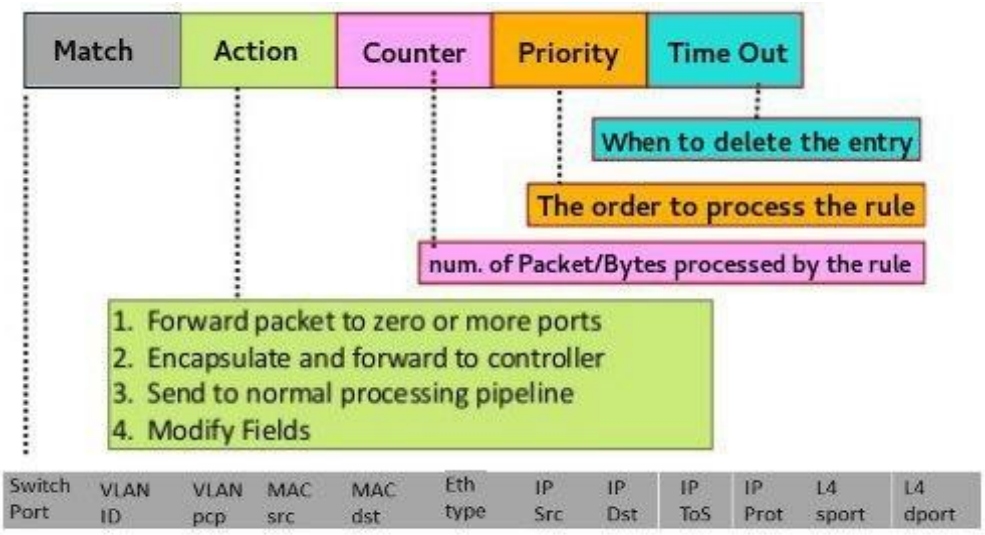
Riguardo alle azioni, quelle tipicamente supportate da uno switch OF sono:

- Output: l'azione di output invia i pacchetti alla porta OF specificata che può essere una porta fisica, logica o riservata.
- Drop: non c'è un'azione esplicita per scartare un pacchetto. In generale, i pacchetti per cui non è specificata un'azione di output dovrebbero essere scartati.

Per l'inoltro del pacchetto si specificano porte che di solito sono porte fisiche dello switch che corrispondono alle sue interfacce hardware ma possono essere anche porte logiche definite dallo switch o riservate. Queste ultime possono indicare un forwarding generico come mandare il pacchetto al controller, effettuare il flooding o per far riferimento a metodi non OF, come il classico forwarding di uno switch. Le porte riservate di solito supportate da uno switch OF sono:

- CONTROLLER: rappresenta il canale verso l'SDN controller. Può essere utilizzata sia come porta di ingresso che come porta di uscita. Quando viene utilizzata come porta di uscita, incapsula il pacchetto in un messaggio Packet-In e lo invia utilizzando il protocollo OF. Se viene usata come porta di ingresso, identifica un pacchetto proveniente dal controller.
- IN_PORT: rappresenta la porta di ingresso del pacchetto.Può essere utilizzata come porta di uscita per trasmettere il pacchetto attraverso la stessa porta da cui è stato ricevuto.
- NORMAL: identifica il normale processamento di forwarding di uno switch L2 non OF e può essere utilizzata come porta di uscita.
- FLOOD: effettua il flooding del pacchetto come un normale switch L2 non OF e può essere utilizzata come porta di uscita e in genere inoltra tutti i pacchetti dalle interfacce fisiche, tranne quella di ingresso da cui è stato ricevuto.[2]

[2] openflow-switch-v1.5.1.pdf (opennetworking.org)

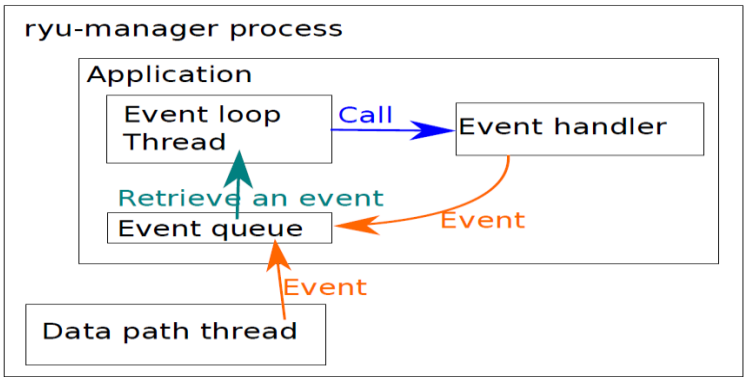


Structure of a flow table entry

In uno switch OF sono presenti due ulteriori tabelle: la Meter table e la Group table ma per questa attività di tesi non vengono utilizzate.

3.1.3 Ryu

Ryu è un SDN controller open-source scritto in Python. Esso fornisce delle APIs che permettono agli sviluppatori di creare nuovi componenti software o di modificare quelli già esistenti per controllare e gestire la rete. Ryu supporta vari protocolli southbound quali OpenFlow, Netconf, OFConfig.



Il modello architetturale e di programmazione di Ryu è basato sul componente Ryu-manager, che rappresenta l'eseguibile principale. Le applicazioni sono classi che ereditano da ryu.base.app_manager.RyuApp e sono "singletone", ovvero solo una singola istanza è supportata. Gli eventi sono classi che ereditano da ryu.controller.event.EventBase. La comunicazione tra le applicazioni avviene trasmettendo e ricevendo eventi. Ogni applicazione ha una sola coda FIFO per ricevere gli eventi. Un thread è creato automaticamente per ogni applicazione e questo thread esegue l'event loop: se c'è un evento nell'event queue, l'event loop lo carica e chiama il corrispondente event handler (questo implica che se l'event handler si blocca, nessun altro evento può essere processato dalla stessa applicazione). Gli event handler sono definiti introducendo il decorator ryu.controller.handler.set_ev_cls sul metodo designato a gestire l'evento.

Ryu si presenta con alcune applicazioni built-in quali:

1. Switching hub: il Ryu controller ci permette di impostare uno switch in grado di imparare i MAC address degli host connessi su una porta e salvarli in una tabella e quando riceve un pacchetto di un host di cui ha gia imparato il MAC address, inviarlo sulla porta opportuna, mentre quando riceve un pacchetto di un host sconosciuto, effettuarne il flooding.
2. Traffic monitoring: allo switching hub viene aggiunta la funzione di monitoring che periodicamente (ogni 10 secondi) invia le richieste FlowStatsRequest e PostStatsRequest agli switch connessi.
3. Switching hub REST: vengono aggiunte due API REST allo switching hub per ottenere la tabella dei MAC address dagli switch ed aggiungere una coppia MAC address/porta alla tabella e una flow entry sullo switch.
4. Link aggregation: funzione di LA per combinare più link fisici in un unico link logico per aumentare la velocità di trasmissione e fornire fault tollerance.
5. Spanning tree: utilizzo dello STP per prevenire il broadcast dei pacchetti in una rete che presenta un loop e utilizzare questa funzione anche come mezzo di recovery per cambiare percorso nel caso in cui ci sia un guasto di rete.
6. Firewall: la funzione di firewall permette di impostare delle regole di firewall sugli switch come flow entries mediante API REST (e.g. per escludere il traffico ICMP tra due host).
7. Router: Il Ryu controller permette di configurare un router mediante API REST per assegnare/modificare gli indirizzi IP, impostare delle rotte statiche e specificare la rotta di default.
8. QoS: possibilità di riservare banda di rete per una comunicazione.

Ulteriori features offerte da Ryu sono una libreria (packet library) che permette di ottenere un oggetto Python dato un pacchetto crudo (e viceversa) e pieno supporto alle versioni del protocollo OpenFlow (possono essere specificati tutti i tipi di match, instructions e actions definiti dalle specifiche di OpenFlow) [RYU].

3.1.4 OpenDaylight

OpenDaylight (ODL) è una piattaforma modulare per automatizzare e gestire le reti anche di grandi dimensioni. Il progetto ODL nasce con il focus sulla programmabilità delle reti ed è composto da 3 livelli:

1. Plugin southbound per comunicare con i dispositivi di rete.
2. Servizi core che possono essere utilizzati per mezzo del SAL (Service Abstraction Layer).
3. Interfacce northbound per permettere agli utenti di applicare policy di alto livello ai dipositivi di rete.

In ODL, tutti i dispositivi di rete e le applicazioni sono rappresentati mediante oggetti o modelli, le cui interazioni sono processate nel SAL. Il SAL è un meccanismo di scambio dei dati adattando il modello YANG di rappresentazione dei dispositivi di rete con quello applicativo. YANG è un modello che fornisce una descrizione generalizzata delle capacità del dispositivo o dell'applicazione.

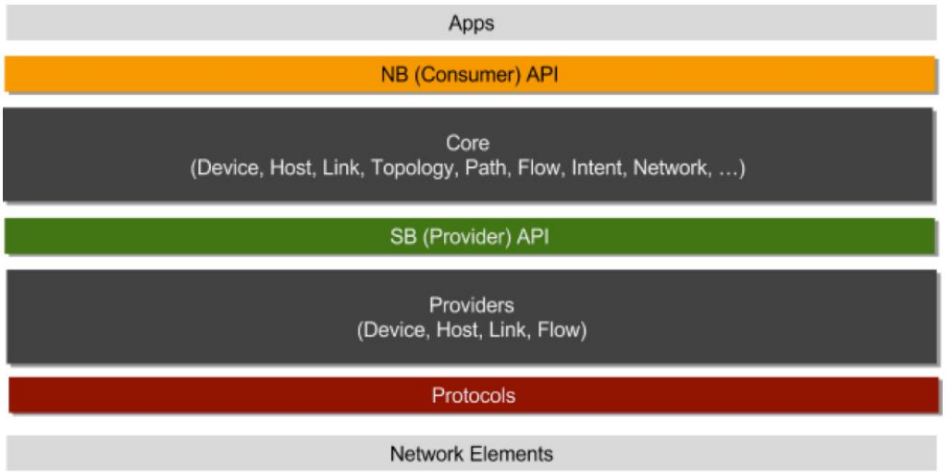
ODL utilizza il framework OSGi per caricare i moduli (o bundle) a runtime, garantendo un approccio flessibile per aggiungere funzionalità e per combinare servizi multipli e protocolli per risolvere problemi più complessi. OpenDaylight include supporto per un ampio set di protocolli quali OpenFlow, OVSDB, NETCONF, BGP e molti altri, così come interfacce northbound come gRPC e RESTful API.

Infine, ODL fornisce fault tolerance e resilienza permettendo a più istanze del controller di esistere e cooperare contemporaneamente: nell'eventualità di un guasto di un'istanza, un nuovo master viene selezionato per prendere il controllo della rete [OpenDaylight].

3.1.5 ONOS

ONOS (Open Network Operating System) fornisce il control plane di una infrastruttura SDN per gestire i dispositivi di rete ed eseguire moduli software per fornire servizi agli host. Il kernel ONOS e i servizi core sono scritti in Java. Esso può essere eseguito come un sistema distribuito per utilizzare le risorse di CPU/memoria di più server e per fornire fault tolerance. Inoltre ha un meccanismo built-in che permette di connettere/disconnettere componenti mentre il controller è in esecuzione (grazie al framework OSGi su cui è costruito).

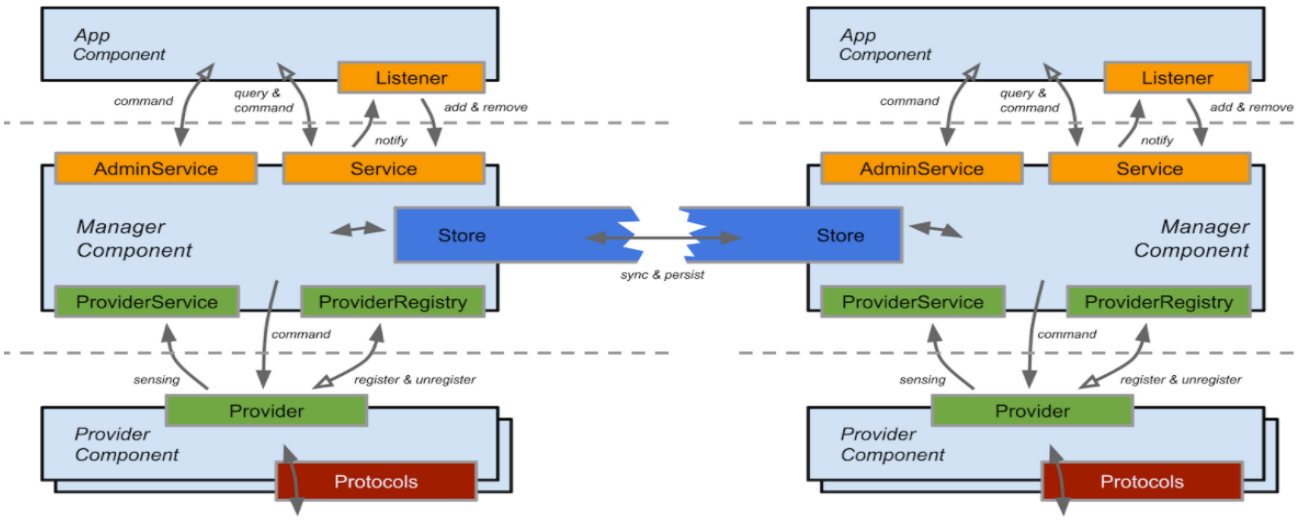
Il design interno dell'architettura di ONOS prevede più strati (l'insieme di tutti i layer viene chiamato System tiers) ognuno con la sua funzionalità.



I servizi che esso offre, anche chiamati subsystem, sono delle unità funzionali composte da più componenti che creano uno spicchio verticale attraverso i tier. I servizi principali definiti da ONOS sono:

1. Device Subsystem: gestisce l'infrastruttura dei dispositivi.
2. Link Subsystem: gestisce l'elenco dei link;
3. Host Subsystem: gestisce le stazioni host e la loro locazione nella rete.
4. Topology Subsystem: fornisce degli snapshot ordinati nel tempo del grafo di rete.
5. Path Subsystem: trova il percorso ottimale tra i dispositivi/host utilizzando l'ultimo snaposhot topologico.
6. FlowRule Subsystem: gestisce le flow entry nei dispositivi di rete e fornisce metriche sui flow.
7. Packet Subsystem: permette alle applicazioni di ricevere i pacchetti provenienti dai dispositivi di rete ed emette pacchetti verso la rete.

Ognuno dei componenti di un subsystem risiede in uno dei 3 tiers principali (Apps - Core - Providers) del System tiers e sono identificati da una o più interfacce Java che essi implementano.



La figura sopra riportata evidenzia le relazioni tra i componenti dei vari subsystem. Le linee tratteggiate rappresentano i confini introdotti dalle API southbound e northbound.

Nel tier più in basso dello stack architetturale, ci sono le interfacce Provider che interagiscono con la rete con librerie protocol-aware e con il core attraverso l'interfaccia ProviderService interface, fornendogli i dati raccolti dal basso.

Il manager è il componente che risiede nel core che riceve informazioni dai Provider e le presenta alle applicazioni o ad altri servizi. Esso espone diverse interfacce:

- "Service", che permette alle applicazioni di conoscere gli aspetti della rete.
- "AdminService", che permette di applicare i comandi provenienti dalle applicazioni sui dispositivi di rete.
- "ProviderRegistry", l'interfaccia attraverso la quale i Provider si registrano al Manager per comunicare.
- "ProviderService", l'interfaccia per permette la reale comunicazione tra Manager e Provider ad esso iscritto.

I consumatori dei servizi del Manager (applicazioni o altri servizi) possono ricevere le informazioni in modo sincrono interrogando il servizio o in modo asincrono utilizzando gli event listener.

Sempre all'interno del core, l'interfaccia Store ha il compito di sincronizzare le informazioni provenienti dal Manager con le altre istanze ONOS comunicando con gli store di tali istanze. Ha inoltre il compito di generare gli eventi (in base alle informazioni ricevute dal Manager). Un evento è caratterizzato da un tipo (e.g. Device) e da un subject. Ad esempio, "DeviceEvent" notifica i DeviceListeners che un dispositivo (subject) è stato eliminato, aggiunto o modificato.

Una volta generato l'evento dallo store, l'evento viene mosso dallo store attraverso l'interfaccia StoreDelegate che a sua volta chiama l'interfaccia EventDeliveryService che si occupa di inoltrare l'evento solo ai listeners interessati.[3]

A livello applicativo, le direttive per i dispositivi di rete sono espresse mediante flow rules. Per alterare in modo più semplice il comportamento della rete, ONOS fornisce un livello di astrazione più alto chiamato

[3] https://wiki.onosproject.org/display/ONOS/Architecture+and+Internals+Guide

“intent” in modo tale che le applicazioni specificano che cosa deve succedere alla rete piuttosto che il come deve avvenire un'operazione. È il framework degli intent (altro non è che un subsystem) a occuparsi di generare internamente la configurazione da iniettare a basso livello.

Esempi di intenti disponibili:

- “HostToHost Intent”: permette di specificare la sorgente e la destinazione (opzionalmente dei criteri di selezione del traffico e come deve essere gestito), successivamente l’intent framework troverà il percorso migliore e installa le flow rules necessarie nel dispositivo. Se un elemento della rete si guasta, l’intent cercherà di trovare un percorso alternativo e generare nuove flow rules. L’intent framework, infatti, è in ascolto per eventuali guasti di rete e cerca di risolverli il prima possibile.
- “PointToPoint Intent”: permette di specificare un percorso alternativo, specificando gli switch del percorso e le relative porte da attraversare (mentre l’HostToHost intent dati due nodi sceglie lo shortest path) [ONOS].

3.1.6 Utilizzo di tecnologie SDN in ambito elettro-energetico e Smart Grid

Questa sezione analizza lo stato dell’arte per quanto riguarda l’utilizzo di tecnologie SDN in ambito elettro-energetico con particolare riferimento alle Smart Grid (SG)[4], presentando le motivazioni che l’hanno portata ad essere adottata in questo ambiente e i vantaggi apportati per migliorare la resilienza, la stabilità e l’ottimizzazione del traffico nelle SG. I concetti qui introdotti sono tratti da [Rehmani2019], che presenta dettagliatamente e in maniera chiara l’area delle SG basate su SDN, includendo riferimenti architetturali, vantaggi/svantaggi e schemi di sicurezza dati dall’introduzione della tecnologia SDN.

La distribuzione elettrica è caratterizzata dalla necessità di un controllo granulare dei parametri fondamentali di funzionamento del sistema (es. potenza erogata), per poter garantire la continuità del servizio attraverso opportune modifiche in real-time del sistema. Questo include la necessità, a livello di infrastruttura ICT, di garantire la comunicazione tra i vari dispositivi coinvolti e poter reagire in maniera appropriata al verificarsi di eventi potenzialmente dannosi (es. mancata connettività con i servizi cloud). Se, da un lato, questa esigenza di comunicazione è alla base del mondo ICT applicate all’ambito elettro-energetico, possiamo evidenziare come nel caso delle SG questa esigenza sia ancora più pressante.

Infatti, lo scopo principale delle SG è quello di ottenere energia elettrica da fonti energetiche rinnovabili e quello che la distingue dalle reti elettriche convenzionali è la sua capacità di effettuare comunicazioni bidirezionali (“demand side management”), ovvero gestire in maniera efficiente i consumi di un sito per ridurre i costi, e *real time pricing* per aumentare la consapevolezza del cliente riguardo al consumo effettuato in brevi periodi di tempo. Le risorse energetiche rinnovabili sono geograficamente distribuite per natura, potenzialmente altamente variabili nel tempo, di vario tipo e generate da tecnologie differenti. Per questo motivo la loro integrazione nelle SG richiede una comunicazione affidabile e a bassa latenza. Esse sono per lo più presenti e connesse all’interno delle abitazioni domestiche intelligenti sotto forma di pannelli solari o

[4] La Smart Grid è una rete elettrica di nuova generazione in cui i processi di gestione e distribuzione dell’elettricità sono progettati incorporando flussi di energia e di informazioni bidirezionali tra fornitori e consumatori, integrando tecnologie digitali avanzate con capacità di elaborazione e pervasività tali da migliorare controllo, efficienza, affidabilità e sicurezza della rete stessa. Una Smart Grid interagisce con gli apparati intelligenti dei consumatori on-premise, con l’obiettivo di utilizzare pienamente le flessibilità disponibili e conseguire obiettivi di risparmio e di efficienza energetica complessiva del sistema, riduzione dei costi e aumento dell’affidabilità in un contesto caratterizzato dalla crescente diffusione di fonti energetiche rinnovabili non programmabili. Essendo una modernizzazione della rete elettrica originaria, soprattutto a livello di distribuzione dell’energia e di gestione delle infrastrutture elettriche attraverso l’integrazione delle moderne tecnologie di comunicazione per scambiare informazioni in tempo reale tra utilizzatori, gestori e fornitori di energia.

piccole turbine a energia eolica e possono immettere notevoli quantità di energia nella rete. Questa immissione deve essere controllata e supervisionata per garantire la stabilità generale del sistema energetico. Inoltre all'interno delle case intelligenti, come si è detto, vengono eseguite varie funzioni per la fatturazione in tempo reale o per l'aggiornamento software dei dispositivi di sensoristica senza l'intervento diretto in loco umano. Anche queste operazioni richiedono un canale comunicativo affidabile e sicuro gestito globalmente.

L'approccio tradizionale adottato nelle prima fasi nelle SG consisteva nell'introduzione di protocolli quali MPLS (Multiprotocol Label Switching) per la comunicazione di sistema in quanto fornisce la possibilità di effettuare traffic engineering e creare reti private virtuali (Virtual Private Networks – VPN), ma non è stato sufficiente: in primo luogo perché l'aggiunta di nuovi servizi nella SG richiederebbe una nuova configurazione dei router ogni volta e questo non è fattibile nel lungo termine, in secondo luogo perché esso obbliga i ricercatori ad effettuare test di nuovi protocolli e servizi implementati direttamente sul campo.

Il paradigma SDN è stato quindi proposto per monitorare e gestire globalmente la comunicazione delle reti, rivoluzionando come questo avveniva precedentemente. L'applicazione di SDN in vari domini non è nuova. Infatti, essa è stata applicata nei data center, nelle wide area network, reti ottiche e reti wireless e può quindi essere adottata ampiamente come supporto nelle comunicazioni in ambienti SG, per migliorare ampiamente l'efficienza, la resilienza di questi sistemi attraverso meccanismi di load balancing, aggiustamento dinamico dei percorsi, rilevamento rapido di guasti, sicurezza e per monitorare il traffico che fluisce al suo interno. Inoltre SDN permette alla SG di evolvere, adottando nuove tecnologie e servizi.

Nell'ambito delle SG, le principali criticità che possono trovare una soluzione con tecnologie SDN sono le seguenti.

1. Resilienza agli attacchi e guasti. I sistemi SG non sono molto resilienti agli attacchi o ai guasti e nell'ipotesi in cui i due eventi si verificano, le SG dovrebbero essere capaci tempestivamente di riconfigurare il loro stato. Di conseguenza, i dispositivi di rete necessitano di essere facilmente programmabili per adattarsi ai cambiamenti avvenuti.
2. Protocolli e dispositivi commerciali. I sistemi SG sono costruiti utilizzando diversi protocolli e dispositivi commerciali, introducendo problemi di interoperabilità. Questo introduce un grado di complessità ulteriore in quanto varie applicazioni vanno implementate ed eseguite per rispondere alla varietà di dispositivi e protocolli.
3. Sicurezza e privacy. Ci sono dei particolari tipi di attacchi quali link flood attack o manipolazione dei dati di monitoraggio che sono specificatamente progettati per deteriorare le performance delle SG.
4. Gestione di rete. La gestione di rete delle SG è complessa, anche dovuto alla numerosità dei componenti coinvolti nonché alla loro distribuzione su scala geografica, richiede un elevato numero di risorse umane (amministratori di rete) se effettuata manualmente con tecnologie tradizionali. Inoltre, l'operatore umano è maggiormente incline ad errori rispetto ad un sistema automatico.

Questi problemi possono essere più agevolmente risolti attraverso tecnologie SDN. Nelle SG si generano diversi tipi di traffico con diversi requisiti di QoS in termini di affidabilità, ritardo e throughput. Per esempio, la fatturazione real-time richiede un'affidabilità maggiore del 98% e un ritardo minore di 1 minuto.

L'SDN controller in questo scenario può identificare il tipo di traffico e prioritizzare dinamicamente il traffico programmando gli switch della rete. Inoltre, esso può decidere quali collegamenti fisici utilizzare in base alle variazioni nelle comunicazioni e grazie alla sua proprietà di indipendenza dai protocolli, risolvere i problemi di interoperabilità nelle SG. In linea di massima, quanto segue riassume le motivazioni dell'impiego del controller SDN in ambiente SG:

- Isolamento dei differenti tipi di traffico generati dai vari dispositivi.
- Nelle SG, dati di misurazione e comandi di controllo necessitano di essere consegnati il prima possibile e richiedono una priorità maggiore rispetto al traffico normale. SDN può aiutare a dare la giusta priorità ai dati sensibili alla latenza; inoltre, avendo una visibilità globale sull'intera rete, l'orchestrazione del traffico viene semplificata.
- Possibilità di creazione di fette di rete virtuali (le cosiddette "slice") in base alla locazione geografica o al dominio considerato. Questo aiuta in quanto ogni fetta ha le proprie policy di sicurezza, gestione e QoS.
- Resilienza nelle SG, ovvero la capacità del sistema di reagire a malfunzionamenti improvvisi o attacchi maliziosi e in risposta a questi, riconfigurarsi mantenendo attivi i servizi forniti.
- Recovery veloce dei guasti. Se un link di comunicazione cade o è congestionato, la SG non funziona correttamente. SDN mitiga questo problema, rilevandolo e risolvendolo quanto prima, ovviamente nei limiti della disponibilità di risorse nell'ambito dell'infrastruttura.
- Interoperabilità, ovvero SDN è una tecnologia che opera attraverso standard aperti, quindi differenti dispositivi di rete possono essere gestiti e configurati.
- Gestione della rete semplificata, in quanto il controller SDN ha visibilità globale della rete e in base ai requisiti delle applicazioni e le condizioni della rete sottostante, esso è in grado di cambiare le regole di processamento dei pacchetti negli switch/router. Senza di esso, si richiederebbe l'intervento manuale per riprogrammare tali apparati.

3.2 Tecnologie per il monitoraggio di informazioni di rete

3.2.1 Netflow

Netflow fu originariamente introdotto da Cisco nel 1995 ma nel corso degli anni esso è diventato uno standard de facto tanto che nel 2008 l'IETF ha rilasciato IPFIX basato sulla versione 9 di Netflow.

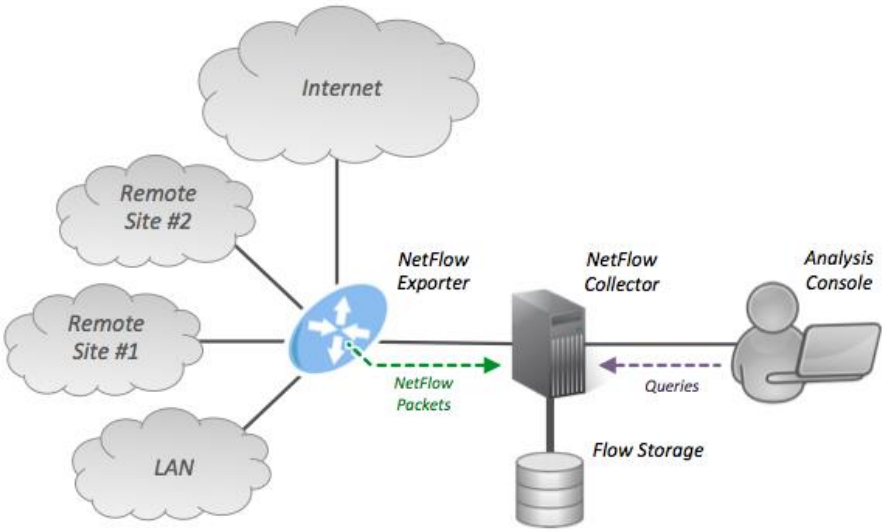
Esso è una funzionalità presente in molti dispositivi di rete che permette di collezionare misurazioni per ogni fusso di rete, identificato dalla caratteristica quintupla (indirizzo IP sorgente/destinazione; protocollo di trasporto; porta TCP/UDP sorgente e destinazione) e di esportarle verso un altro sistema per analisi da svolgersi in un intervallo temporale successivo.

Il funzionamento di Netflow si basa sul fatto che la stragrande maggioranza delle comunicazioni end-to-end tra dispositivi IP avviene attraverso sessioni di livello 4 della pila protocollare ISO/OSI (livello di trasporto), attraverso i protocolli di comunicazione TCP (Transmission Control Protocol) oppure UDP (User Datagram Protocol). In termini tecnici, un flow è definito dalla 5-tupla (una collezione di 5 valori) seguente:

- Indirizzo IP sorgente e di destinazione coinvolti nello scambio delle informazioni;
- Porte sorgenti e di destinazione (se ce ne sono);
- Il protocollo.

Tutti i pacchetti che condividono la stessa 5-tupla appartengono quindi allo stesso flusso.

Analizzando e collezionando i dati dei flussi possiamo conoscere i dettagli di come la nostra rete viene utilizzata (il numero totale di pacchetti e byte scambiati, ad es. utile per informazioni di fatturazione, il timestamp del primo e dell'ultimo pacchetto del flow, quindi la sua durata, ecc..).



Per utilizzare Netflow è necessario disporre di 3 componenti:

- Flow exporter (di solito il router) che colleziona le informazioni sui flow e le invia al flow collector;
- Flow collector (di solito un server) è un datastore dei flow esportati;
- Flow analyzer, un'applicazione che analizza le informazioni immagazzinate nel flow collector.

I flow vengono esportati periodicamente al flow collector in un processo chiamato flow expiration. Una specifica flow entry scade al verificarsi di due eventi:

1. Inactive timeout: se il flow è rimasto inattivo, ovvero nessun pacchetto è stato osservato per un lasso di tempo (ad es 5 minuti), si assume che il flow ha terminato e la sua entry scade.
2. Active timeout: se il flow rimane attivo per un certo periodo di tempo, esso scade. Questo secondo evento è necessario per poter inviare informazioni in tempo reale al flow collector anche per quanto riguarda flussi molto lunghi temporalmente, evitando che la visibilità di questi flussi sia disponibile solamente quando il flusso stesso termina.

Una volta che il flow collector riceve una flow entry, esso lo invia al flow analyzer che estrae le informazioni più rilevanti dai dati ricevuti quali:

- il traffico diviso per applicazione, protocollo, dominio, indirizzo IP sorgente/destinazione e porta sorgente/destinazione;
- gli indirizzi IP più utilizzati così come i flow più utilizzati;
- La geolocalizzazione degli indirizzi IP sorgente/destinazione.

In generale, il grado di sofisticatezza di processamento del Flow Analyzer è variabile a seconda delle implementazioni. A livello base, possiamo sicuramente dire che queste informazioni ci aiutano a capire se la rete sta avendo dei rallentamenti e chi/cosa li sta causando, chi sta utilizzando una determinata applicazione, quante connessioni sta ricevendo un server, ecc.. Con opportuni algoritmi, infatti, è possibile anche determinare comportamenti anomali dei nodi (sia i dispositivi di rete, che i nodi terminali), eventuali problemi di sicurezza, e molto altro.

3.2.2 SNMP

SNMP (Simple Network Management Protocol) è un protocollo applicativo dello standard TCP/IP utilizzato per descrivere i dati dei dispositivi gestiti nelle reti IP, gestirne lo scambio tra vari sistemi eterogenei, collezionarli e modificarli per cambiare il comportamento dei dispositivi.

SNMP è molto utilizzato nel network management per il monitoraggio della rete. Il protocollo include la descrizione dei dati di management in forma di variabili chiamate MIB (Management Information Base) che descrivono lo stato del sistema e la sua configurazione e le operazioni per richiedere i dati e modificarli.

Nello scenario classico in cui si utilizza SNMP, dei computer chiamati managers hanno il compito di monitorare e gestire un gruppo di dispositivi presenti in una rete. Ogni dispositivo gestito dal manager esegue localmente un agente che riporta le informazioni in formato SNMP al manager.

I componenti coinvolti quindi in un ambiente SNMP sono i seguenti:

1. MIB: SNMP espone i dati di management dei dispositivi gestiti in forma di variabili, organizzate seguendo una struttura gerarchica. In sé il protocollo non definisce quali MIB il dispositivo deve offrire, ma utilizza un design estensibile che permette alle applicazioni di definire la propria gerarchia. Ci sono molti MIB, ognuno specifico per un particolare tipo dato e della struttura ad albero che lo definisce, ogni ramo rappresenta una classificazione del dato mentre le foglie uno specifico dato. Ad ogni ramo è associato un numero e questi possono essere concatenati tra di loro in notazione decimale con punti per creare un identificativo univoco chiamato OID (Object ID).
2. Agent: esso rappresenta il software che viene eseguito sui dispositivi che ha conoscenza locale sui dati di management. SNMP non è necessariamente conforme a un'architettura client/server ma se volessimo forzarlo, l'agent prenderebbe il ruolo del server, responsabile di gestire i MIB, rispondere alle richieste del manager inoltrando i dati e generare le trap.
3. Manager: controlla e monitora i dispositivi SNMP e in un'architettura client/server esso ha il ruolo del client, dovendo generare le query agli agent per ottenere le informazioni desiderate, ascoltare le trap provenienti da essi e riporta in un formato desiderato i dati ottenuti.

Per quanto concerne le interazioni tra i vari componenti, SNMP definisce 7 PDU (Protocol Data Unit) indentificati dal campo PDU-type, e sono: GetRequest, SetRequest, GetNextRequest, GetBulkRequest, Response, Trap ed InformRequest.

3.2.3 SNMP vs. Netflow: differenze

SNMP ci permette di monitorare un gran numero di oggetti (OID) non necessariamente legati al traffico di rete come, ad esempio, la percentuale di utilizzo della CPU, il consumo di RAM e disco (cosa che Netflow non ci permette di fare) e parametri di rete quali il numero di pacchetti persi per porta e la banda utilizzata.

D'altro canto, Netflow è nato con lo scopo di monitorare il traffico di rete e fornisce un dettaglio maggiore su cosa sta succedendo nella rete. Infatti, SNMP non è in grado di ottenere informazioni quali l'host sorgente, il tipo di traffico generato (ovvero che cosa sta facendo l'host) e la destinazione.

In definitiva, le due soluzioni offrono approcci diversi e andrebbero combinati insieme: SNMP da un lato per monitorare la performance dei dispositivi e prevenire guasti hardware, Netflow dall'altro per ottenere un'elevata accuratezza nell'analisi del traffico di rete.

In ambedue i casi, tuttavia, l'approccio prevalente al loro utilizzo è quello del monitoraggio, non dell'"enforcing" di configurazioni. Per quanto riguarda Netflow, questo è evidente dalla sua natura. Per quanto riguarda SNMP, anche se lo standard prevede la possibilità di essere usato in modalità di

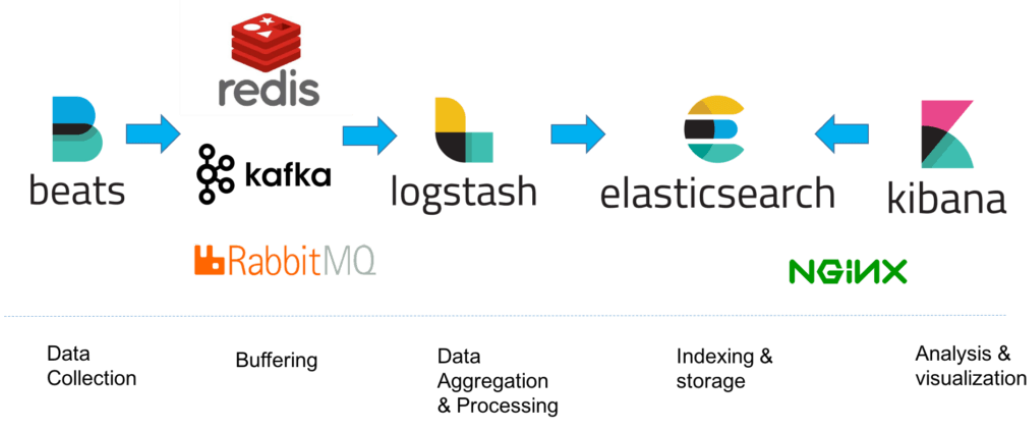
configurazione degli agenti remoti, questo non viene tipicamente utilizzato nel mondo industriale principalmente per problemi di sicurezza e di scalabilità. Pertanto, in reti gestite da SNMP, la configurazione degli apparati viene tipicamente effettuata con meccanismi alternativi, ad es attraverso la configurazione manuale ("Command Line Interface" – CLI) degli apparati.

3.3 Tecnologie per il monitoraggio a livello applicativo: lo stack ELK

Il prodotto più utilizzato per il monitoraggio di informazioni a livello applicativo è la pila di tecnologie che va sotto il nome di "ELK Stack". ELK è un acronimo che identifica tre progetti open source: Elasticsearch, Logstash, Kibana. Essa è utilizzata per il monitoraggio e l'analisi dei dati e permette di riceverli in qualsiasi formato, generati da qualsiasi sorgente, per poi salvarli in un datastore per successive ricerche, analisi e visualizzazioni real-time.

Elasticsearch viene utilizzato come datastore e come motore di ricerca, Logstash per aggregare i dati e Kibana come tool grafico di visualizzazione/analisi. Molto spesso ELK viene utilizzato in congiunzione ad un quarto elemento, Beat, un data-shipper installato sui client che colleziona i dati per poi inviarli ad Elasticsearch/Logstash (con l'introduzione di Beat il nome del prodotto è passato da ELK a ELK Stack).

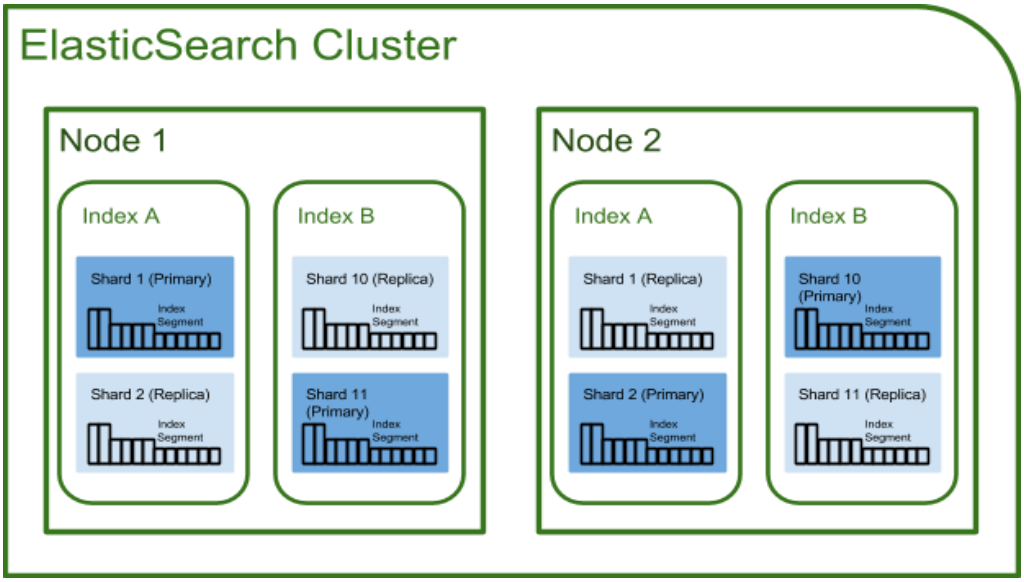
Ciascun tool della pila può essere configurato editando l'apposito file con estensione .yml presente nella cartella di installazione [Elastic].



3.3.1 Elasticsearch

Elasticsearch è il cuore della pila ELK, essendo lo storage di tutti i dati. Prima di analizzarlo è importante definire e capire i termini utilizzati per identificare gli elementi concettuali che lo compongono:

- Cluster: un cluster consiste in uno o più nodi ed è identificato dal suo nome;
- Node: un nodo è una singola istanza di Elasticsearch, di solito eseguita in un box separato/virtual machine dedicata;
- Index: un indice è una collezione di documenti con caratteristiche simili;
- Shard: un indice è diviso in elementi chiamati shard che sono distribuiti tra più nodi e ogni shard può avere zero o più repliche. Di default, un indice è creato con 5 shard ed 1 replica per shard.



Elasticsearch è un datastore NoSQL (ovvero non c'è la necessità di definire uno schema a priori prima di inserire i dati) distribuito. I dati sono salvati sotto forma di documenti JSON. Il fatto di essere distribuito significa che è possibile aggiungere server (nodi) al cluster per aumentarne la capacità ed automaticamente Elasticsearch distribuisce i dati e l'overhead delle query tra tutti i nodi disponibili. Questo meccanismo fornisce scalabilità ed alta disponibilità dei dati. Inoltre, distribuendo i documenti in indici tra diversi shard, e distribuendo questi shard tra vari nodi, Elasticsearch fornisce ridondanza e protegge da guasti hardware. Quando il cluster aumenta/diminuisce in dimensione, Elasticsearch automaticamente migra gli shard per riequilibrare il cluster.

Per interrogare ed interagire con Elasticsearch, vi sono a disposizione delle APIs REST con cui è possibile effettuare le classiche operazioni CRUD (Create, Read, Update, Delete) sul cluster.

3.3.2 Kibana

Kibana è la piattaforma di analisi e visualizzazione dei dati contenuti in Elasticsearch. La sua potenza sta nella libertà fornita all'utente di selezionare la forma grafica da dare ai dati. Kibana offre la possibilità di ricercare i dati e di visualizzarli sotto forma di grafi, mappe, tabelle, semplificando notevolmente l'analisi dei dati, spesso presenti in volume elevato. La dashboard combina questi elementi grafici in un'unica vista accessibile via browser, per fornire le analisi in tempo reale in supporto ai seguenti casi d'uso:

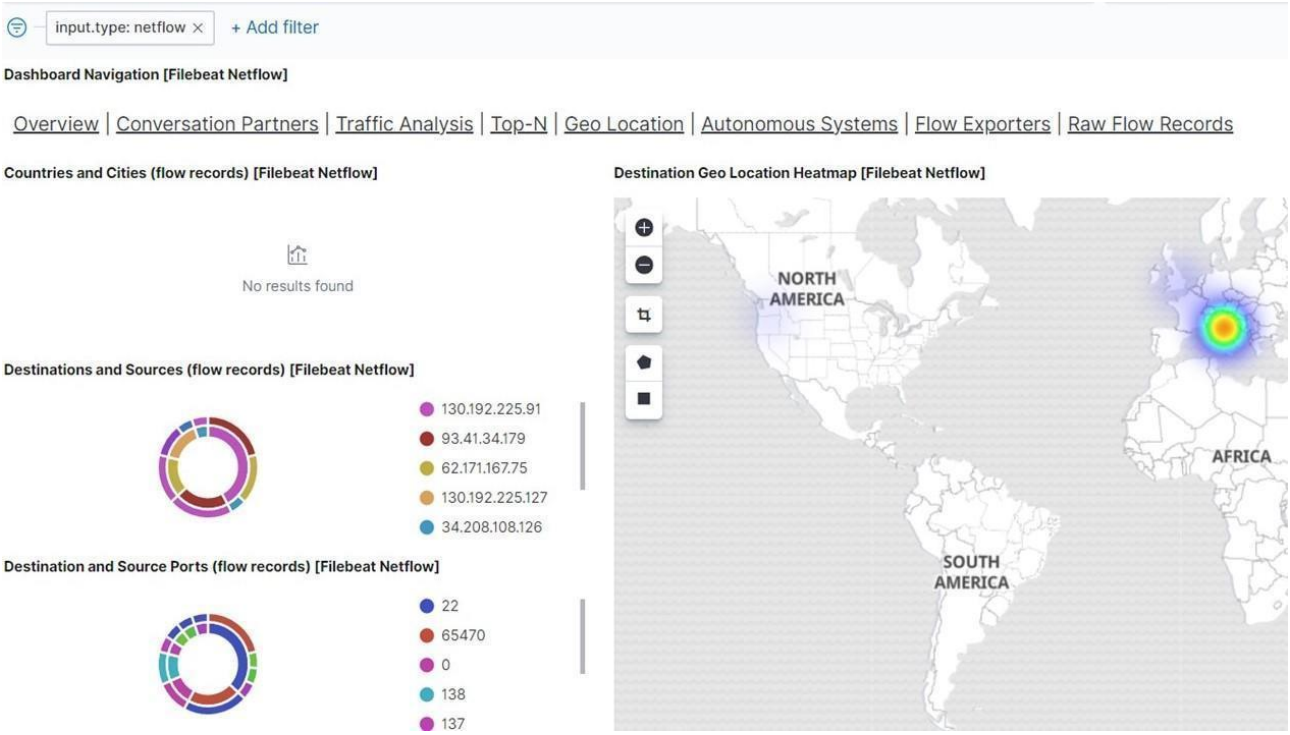
- Analisi dei file di log;
- Monitoraggio di container, delle metriche infrastrutturali e delle performance delle applicazioni;
- Visualizzazione e analisi di dati geospaziali;
- Analisi di sicurezza e di business.

Kibana rappresenta, grazie all'interfaccia web, il punto di accesso centralizzato per impostare i parametri di sicurezza, gestire e monitorare l'istanza ELK. La user interface (UI) di Kibana permette all'utente di accedere visivamente ai dati contenuti negli indici di Elasticsearch. Gli indici sono creati quando Logstash o i Beat inviano dati non strutturati da file di log o da altre sorgenti e li convertono in un formato strutturato per abilitare le funzionalità di salvataggio e di ricerca di Elasticsearch.

Le dashboard di Kibana sono costituite da più "Visualize". Un visualize permette di creare delle visualizzazioni dei dati presi dagli indici in Elasticsearch in forma di mappe, pie chart, tabelle, grafi.

Le visualize di Kibana sono sono basate sulle query di Elasticsearch. Elasticsearch infatti fornisce delle API REST che sono usate dalla componenti grafiche e possono essere chiamate per configurare ed accedere alla features di Elasticsearch.

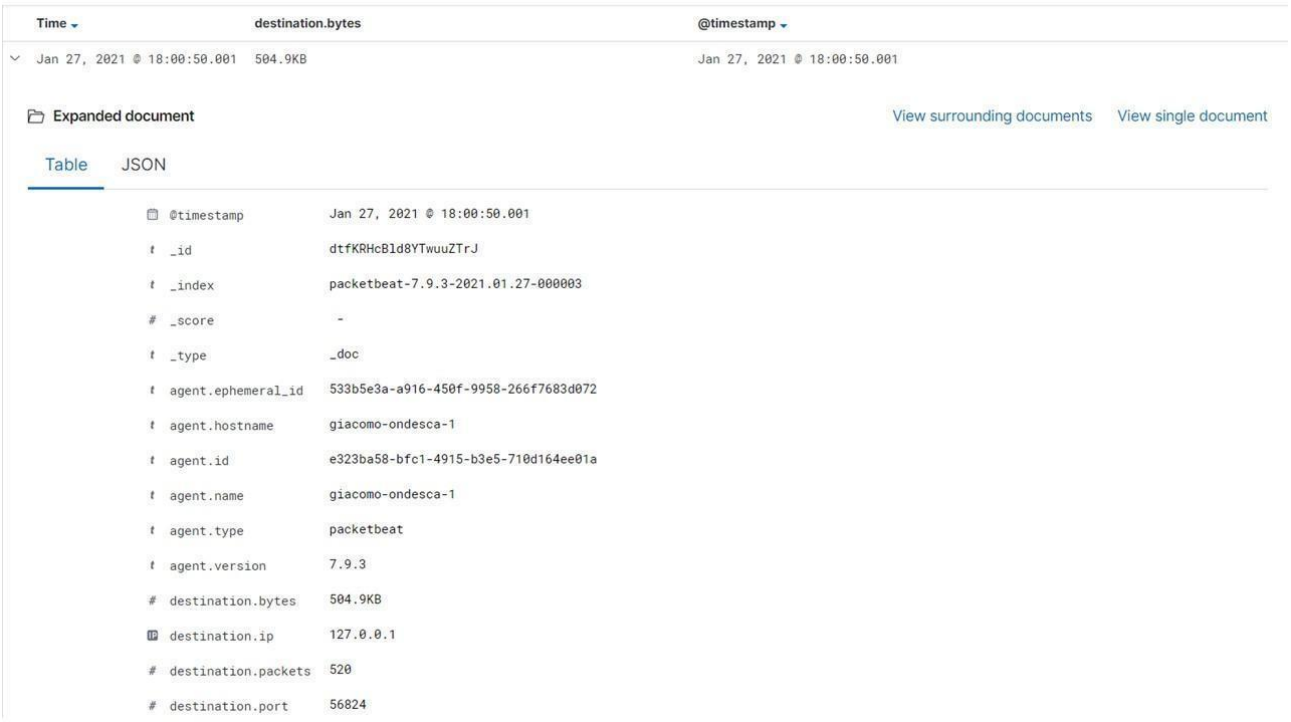
La figura sottostante riporta la visualize Geo-Location della Dashboard relativa al modulo Netflow di Filebeat.



Una delle prime tab della UI di Kibana utilizzate per cercare, filtrare e poi visualizzare i dati è chiamata Discover. La prima cosa da fare è specificare l'indice dove sono presenti i dati che ci interessano e l'intervallo temporale (di default vengono recuperati i dati degli ultimi 7 giorni). Discover include una tabella che mostra tutti i documenti e contiene delle colonne per il campo temporale di indicizzazione e il documento "_source", ovvero il campo che contiene il corpo del documento JSON originale, che può esse quindi grande in dimensione e per numero di campi. Si ha quindi la possibilità di specificare un sottoinsieme dei campi da visualizzare nelle colonne della tabella. Inoltre, una delle capacità uniche di Discover è l'abilità di combinare ricerche testuali e filtri basati sui dati strutturati per definire un sottoinsieme dei documenti a cui si è interessati, per filtrare i risultati ed escludere/includere dei campi o in base a un range di valori, e molto altro. Per effettuare query più complesse si utilizza Kibana Query Language (KQL). Posizionando il cursore sulla

barra di ricerca, non appena si inizia a digitare dei caratteri da tastiera, vengono mostrati suggerimenti per campi, valori e operatori da utilizzare per la query. Discover ritorna al massimo 500 documenti che sono utilizzati per calcolare i valori più popolari per un determinato campo.

C'è la possibilità di analizzare ogni singolo documento per leggerne i campi, cliccando sulla freccia in basso a inizio di ogni riga per espandere il documento. Per visualizzare i documenti che sono stati indicizzati prima e dopo di esso, cliccare su "View surrounding documents", mentre per effettuare l'accesso diretto al documento, cliccare su "View single document".



3.3.3 Logstash

Logstash permette di collezionare log da varie sorgenti (utilizzando gli input plugin), processare i dati in un formato comune utilizzando i filtri per verificare se l'evento soddisfa dei criteri ed effettuare lo stream dei dati a vari output (per l'analisi verso Elasticsearch, ma non è l'unica punto di destinazione ammesso) utilizzando gli output plugin.

La pipeline di processamento dell'evento in Logstash ha quindi tre fasi eseguite nell'ordine qui riportato:

1. Input plugin: permette a una specifica sorgente di un evento di essere letta da Logstash. Gli input plugin più utilizzati sono il plugin *file*, per leggere da un file in un filesystem, il plugin *redis*, per leggere da un server redis e il plugin *beats*, per processare gli eventi inviati dai Beat;
2. Filter plugin (opzionale): esegue un processamento intermedio dei dati. Si possono combinare più filtri in cascata, specificando la condizione di azione del filtro se l'evento soddisfa alcuni criteri. Alcuni dei filtri più utilizzati sono:
    - *grok*, per analizzare e strutturare dei file di testo. Grok è la scelta ideale attualmente in Logstash per analizzare dati di log e per trasformarli in dati strutturati, disponibili per essere indicizzati e ricercati mediante query.
    - *mutate*, per effettuare trasformazioni generiche sui campi dell'evento. Si ha la possibilità di rinominare, rimuovere, sostituire o modificare i campi dell'evento.
    - *drop*, per scartare completamente un evento.
    - *clone*, per realizzare una copia dell'evento, con la possibilità di aggiungere o rimuovere campi.
    - *geoip*, per aggiungere informazioni sulla locazione geografica degli indirizzi IP.
3. Output plugin: Questa è la fase finale della pipeline di Logstash e permette di inviare i dati a una differente destinazione. Un evento può passare attraverso molteplici output e una volta completati tutti i processamenti degli output, l'evento termina la sua esecuzione. Gli output plugin utilizzati includono *elasticsearch*, per inviare dati ad Elasticsearch, il plugin *file*, per scrivere i dati dell'evento su un file su disco, *graphite*, per inviare i dati a Graphite, un tool open-source utilizzato per il salvataggio e la visualizzazione grafica di metriche, e molti altri.

3.3.4 Beat

I Beat sono data-shipper open-source installati come agenti sui server/dispositivi per collezionare metriche e log ed inviarli a Elasticsearch. Nella pila ELK tradizionale, l'aggregazione ed il processamento erano compiti destinati a Logstash, ma per come esso era stato progettato, in presenza di pipeline complesse che richiedevano un'elevata capacità computazionale, la performance non era delle migliori. Per questo motivo il compito di estrarre i dati viene affidato ad un ulteriore tool, appunto Beat.

Esistono molti Beat presenti in Elastic che permettono di collezionare vari tipi di metriche, i più utilizzati sono:

- Filebeat: utilizzato per inviare i log ad Elasticsearch. Esso consiste di due componenti principali: gli harvester e gli input. I primi sono responsabili di leggere il contenuto di un singolo file. L'harvester legge un file, linea per linea, e manda il contenuto verso l'output. Si ha un harvester per ogni file ed esso ha la responsabilità di aprire e chiudere il file. I secondi sono responsabili di gestire gli harvester e trovare le sorgenti dei file contenenti i log da leggere. L'input cerca tutti i file che soddisfano i glob path specificati ed esegue un harvester per ogni file trovato. Filebeat è integrato con dei moduli che

semplificano la collezione, il parsing e la visualizzazione dei formati log più comuni (es. Netflow module).

- Packetbeat: protocolli di rete a livello applicativo (come HTTP) permettono di farci un'idea su come il nostro sistema viene utilizzato: tempo di risposta, errori e latenze delle applicazioni, pattern di accesso dei vari utenti e molto altro. Packetbeat quindi è un analizzatore real-time di pacchetti di rete, permette di accedere a questi dati per capire come il traffico sta fluendo nella rete, di monitorare le applicazioni e le performance del sistema. Esso supporta molti protocolli applicativi (DNS, HTTP, REDIS, ICMP, DHCP ecc.…) ed è possibile specificare su quali mettersi in ascolto sul file di configurazione *packetbeat.yml* nella sezione *protocols.*

Esso decodifica il protocollo a livello 7, successivamente correla la richiesta con la relativa risposta (chiama transazione) e per ogni transazione Packetbeat inserisce un documento JSON in Elasticsearch, indicizzando i campi più interessanti e pertinenti di ognuna.

In Packetbeat (così come in Filebeat e in ogni altro Beat) c'è la possibilità di introdurre i "processors". Questi sono utili per filtrare ed arricchire i dati prima di inviarli all'output specificato. Filtrare i dati in particolare risulta fondamentale per non indicizzare in Elasticsearch tutti i dati di rete, ma solamente quelli di reale interesse, diminuendo la mole di informazioni da salvare ed analizzare. Per definire un processor bisogna specificare il nome del processor (il nome stesso identifica l'azione da eseguire), la condizione che determina quando effettuare l'azione e una serie di parametri opzionali. I processors possono essere definiti a livello globale, quindi applicati a tutti gli eventi gestiti da Packetbeat, oppure a livello di singolo protocollo applicativo.

Alcuni esempi di processors:

- add_host_metadata: annota ogni evento con metadati relativi alla macchina host (es. nome della macchina, versione del sistema operativo, etc);
- include_fields: indica quali campi esportare se una determinata condizione (opzionale) si verifica;
- drop_event: elimina del tutto un evento al verificarsi di una condizione (obbligatoria altrimenti tutti gli eventi vengono eliminati).

Di particolare importanza è il processor che permette di aggiungere metadati riguardanti la locazione geografica degli indirizzi IP, basandosi sui dati presi dai database di Maxmind, chiamato GeoIP processor. Questo aggiunge le informazioni sotto il campo geoip e risolve sia indirizzi IPv4 sia indirizzi IPv6 e non è più distribuito come plugin, ma dalla versione di Elasticsearch 6.7.0 viene rilasciato come un modulo incluso di default in Elasticsearch, chiamato ingest-geoip. Esso ha per default i database GeoLite2 City, GeoLite2 Country, GeoLite2 ASN di Maxmind, da cui attinge per aggiungere i metadati ai record.[5]

3.4 Soluzioni host-based: eBPF

Il kernel Linux è sempre stato il luogo ideale per implementare soluzioni per il monitoraggio, networking e sicurezza di sistema. Linux è suddiviso in due aree distinte: kernel space e user space. Il primo è dove il codice del Sistema Operativo risiede e ha pieno accesso alle risorse hardware (memoria, CPU, ecc…). Data la sua natura privilegiata di accesso a queste risorse, questo spazio è protetto e non permette di eseguire codice non fidato. Lo user space include tutto il codice che vive al di fuori del kernel e i programmi qui eseguiti hanno

[5] https://www.elastic.co/guide/en/elasticsearch/reference/master/geoip-processor.html

accesso limitato alle risorse hardware e questo può avvenire attraverso chiamate di sistema le cui API sono esposte dal kernel (attraverso le cosiddette *system calls*).

Tuttavia, in tempi recenti sono emerse due esigenze non soddisfatte dal kernel Linux.

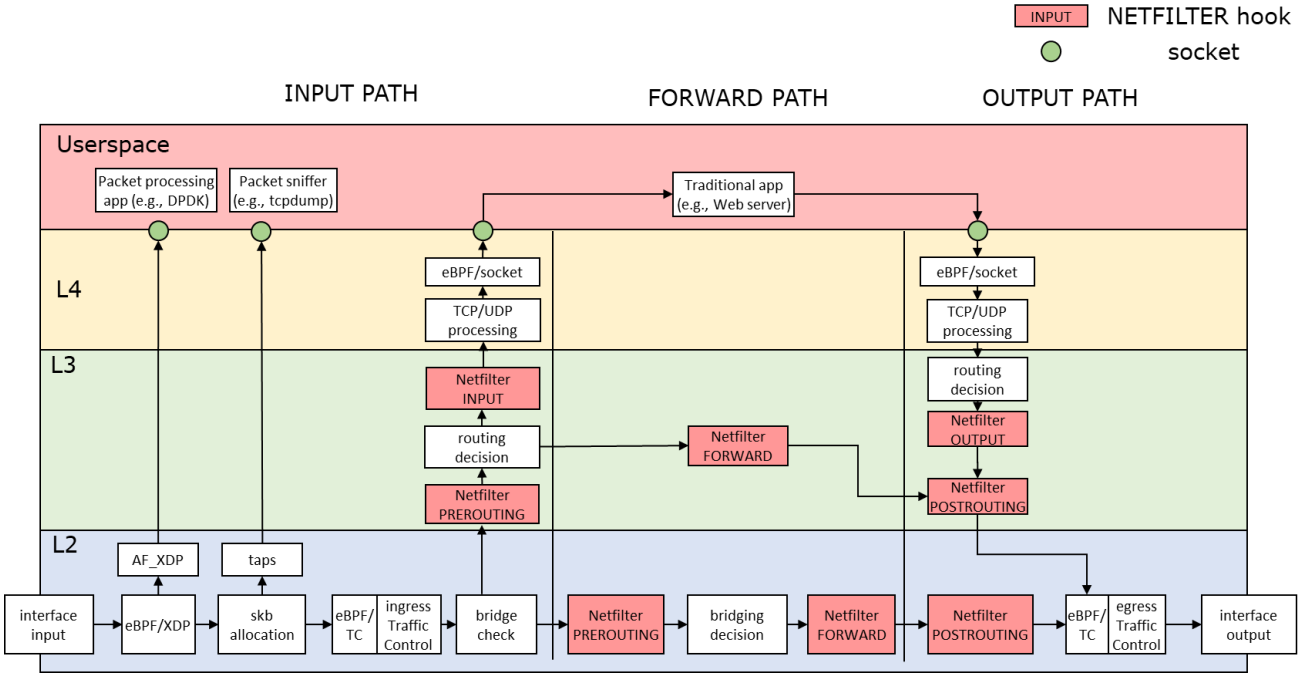
- In particolare per gli aspetti di networking, viene ritenuto necessario poter personalizzare a run-time il processamento di un pacchetto, andando al di là dei classici bridge/router (e poco più) implementato nel kernel, senza dover necessariamente ricompilare il kernel stesso, reinstallarlo e procedere ad un reboot del sistema.
- Le capacità di monitoraggio del funzionamento del sistema (es. il numero di chiamate di una certa funzione kernel, nonchè i valori dei parametri con i quali viene chiamata) erano limitate, rendendo poco efficace lo sviluppo di strumenti in grado di tenere sotto controllo il funzionamento del sistema.

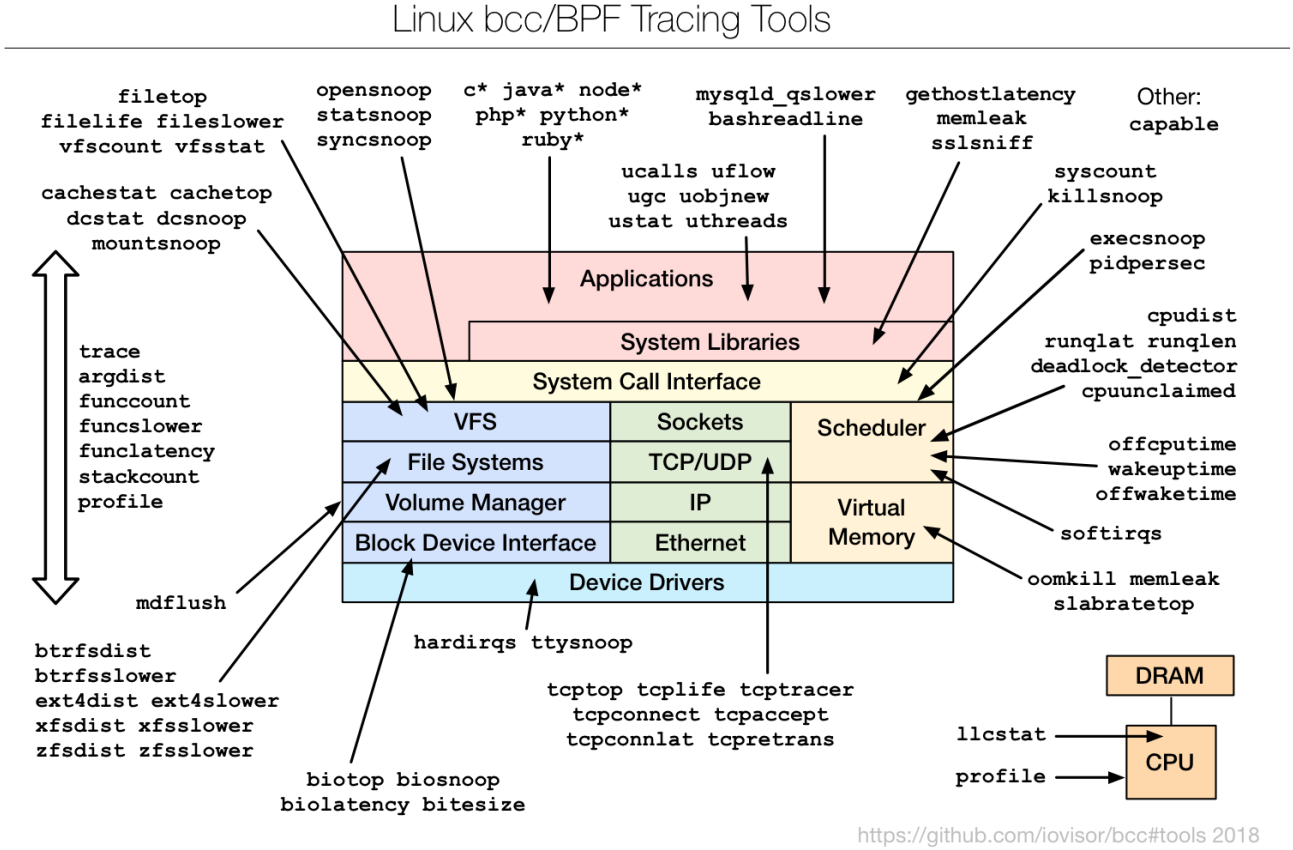
Ambedue le esigenze possono essere soddisfatte da un meccanismo in grado di creare del codice "custom" e iniettarlo dinamicamente nel kernel, senza però aggiungere codice direttamente nel sorgente del kernel stesso. eBPF (extended Berkeley Packet Filter) è pertanto una tecnologia che permette di iniettare ed eseguire programmi protetti da un sandbox (che garantisce la sicurezza dei programmi iniettati) direttamente nel kernel Linux senza dover modificare il codice sorgente o caricare moduli aggiuntivi. Questo rende il kernel programmabile e il nuovo software può sfruttare i layer esistenti e arricchirli con nuove feature senza aggiungere maggior complessità al sistema con l'introduzione di ulteriori layer.

eBPF ha permesso lo sviluppo di una nuova generazione di tool nell'aree del networking, sicurezza, monitoraggio applicativo e delle performance che non si affidano più alle funzionalità kernel ma ne riprogrammano attivamente il comportamento a runtime senza comprometterne l'efficienza di esecuzione e la sicurezza, grazie appunto al sandbox. Prima di essere caricato all'interno del kernel, il programma eBPF deve soddisfare una serie di requisiti. Questa verifica è effettuata da un componente chiamato "verifier" che esegue virtualmente il programma eBPF andando a considerare tutti i casi possibili di esecuzione che si possono ottenere, controllando che questo completi la sua esecuzione, non avendo quindi loop al suo interno, e che rispetti altre specifiche quali stato valido dei registri, dimensione appropriata del programma o che abbia jump a locazioni di memoria fuori dal suo dominio di azione.

Esistono numerosi strumenti di sviluppo che facilitano l'implementazione e la gestione di programmi eBPF. Per esempio, BCC (BPF Compiler Collection) è un framework che permette agli utenti di creare programmi Python con programmi eBPF integrati al loro interno. Questo framework è ideale per i casi d'uso che riguardano il profiling e il tracing del sistema e delle applicazioni dove i programmi eBPF sono utilizzati per collezionare statistiche o generare eventi, la cui controparte in user space permette di visualizzarli in un formato leggibile all'utente. L'esecuzione dei programmi Python genera il bytecode eBPF e lo carica direttamente nel kernel.

A seguire viene riportata l'architettura di eBPF per quanto riguarda il networking, con l'indicazione delle posizioni in cui è possibile eseguire codice iniettato all'interno del kernel di Linux. Inoltre, vengono riportati un elenco dei principali strumenti di monitoraggio open-source relativamente a varie porzioni del kernel.





3.5 Impiego delle tecnologie sui dispositivi

Le varie tecnologie presentate nelle sezioni precedenti richiedono requisiti differenti in termini di risorse hardware per poter funzionare correttamente. Per Elasticsearch ad esempio ci sono delle raccomandazioni

da considerare quando lo si mette in campo, specie nella fase di produzione (ovviamente tutto dipende dal contesto in cui lo si impiega, ma ci sono dei requisiti minimi che è opportuno rispettare). La memoria RAM è la prima risorsa ad esaurirsi dato che le aggregazioni e gli ordinamenti ne richiedono molta. L'ideale sarebbe avere una macchina con 64 GB di RAM, ma anche il taglio 16/32 GB è comune, mentre scendere sotto gli 8 GB è controproducente (porta ad avere cluster con molte macchine). In termini di CPU, Elasticsearch non pone grandi vincoli essendo il suo utilizzo moderato (un processore moderno multicore è più che sufficiente). Per quando riguarda invece lo spazio su disco, esso risulta molto importante per tutti i cluster che indicizzano molti dati. Inoltre, in molti sistemi questo risulta essere il componente più lento e se i cluster scrivono massivamente su disco, la probabilità di saturarlo è molto alta. Per questo motivo è opportuno utilizzare degli SSD. Ultimo ma non per peso, consideriamo le risorse di rete. Data la natura distribuita di ELK, una latenza piccola permette ai nodi di comunicare facilmente mentre una banda elevata ci aiuta a replicare le shard ed effettuare un recovery nel più breve tempo possibile (1GbE/10 GbE è sufficiente per la maggior parte dei cluster).

Per questi motivi Elasticsearch, così come Kibana e Logstash si prestano ad essere installati su macchine/server con i requisiti elencati, quindi non possono essere impiegati in dispositivi embedded o in dispositivi di rete che di solito hanno risorse inferiori di quelle necessarie.[6]

Un discorso diametralmente opposto si verifica per le tecnologie Beat e SNMP. L'ultimo spesso è già presente in molti dispositivi di rete (e non solo) e non richiede grandi risorse hardware; stesso discorso per i Beat, inoltre il fatto di poterli containerizzare (vedere sezione 5) ci permette di impiegarli in ogni dispositivo.

Per Netflow, la funzione di flow export è svolta dal router, mentre per il flow collector sono necessari una buona capienza di storage e per l'applicazione che analizza i dati almeno una CPU quad-core e 4 GB di RAM. Questo implica che sia meglio evitare i dispositivi embedded come componenti della sua architettura.

Il tutto è riepilogato nella tabella seguente.

	NetFlow	SNMP	ELK	Beats
Embedded device	✗	✓	✗	✓
Host	✓	✓	✓	✓
Dispositivi di rete	✓	✓	✗	✓

3.6 Volume dati

Prima di impiegare delle tecnologie su un sistema è bene effettuare una stima dell'overhead sul traffico di rete introdotto da queste. Ovviamente definire il numero esatto del volume dati generato non è semplice dato che esso dipende dal pattern di traffico specifico del nostro ambiente. Per il flow exporter ad esempio la banda utilizzata per esportare i flow al collector è determinata da alcuni fattori quali i valori dell'active/inactive timer, la dimensione/utilizzo della cache, il numero dei flow e la durata stessa del flow, il tutto moltiplicato per la lunghezza di un record Netflow che è di circa 300 byte. In generale, se abbiamo molti flow ma di breve durata, la frequenza di esportazione è dettata dall'inactive timeout, viceversa se abbiamo pochi flow ma di lunga durata (e una dimensione della cache tale da poterli contenere), la frequenza di esportazione è dettata dell'active timeout.[7]

[6] https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html

[7] https://community.cisco.com/t5/other-network-architecture/how-much-bandwidth-netflow-export-uses/td-p/2659292

Per SNMP, il traffico generato effettuando il poll su una singola interfaccia del dispositivo di rete con periodicità di 1 secondo è di circa 6Kb in un minuto (o circa 100bps, anche se il risultato più attendibile è 6Kbps dato che la rete vede quel traffico come un singolo burst di dati).[8]

Per la pila ELK, l'overhead introdotto dai Beat installati sui dispositivi di sensoristica è trascurabile (così come spiegato nella documentazione ufficiale di Elastic). Per Elasticsearch, stimare il carico di lavoro e i requisiti minimi di storage dipendono dal numero di log generati dalle applicazioni e per quanto tempo questi dati devono essere conservati. Inoltre, per il dimensionamento dello storage bisogna considerare il numero di repliche per index e il suo overhead (un indice è circa il 10% più grande della sorgente dati che deve contenere) e lo spazio riservato al sistema operativo.[9]

NetFlow	SNMP	ELK Stack
Generalmente 300 byte per record (oppure 1,5% della banda per interfaccia monitorata).	Circa 6000 bits interrogando un'interfaccia per un minuto (100 bps).	Dipende dai log generati dalle applicazioni. Solitamente vengono immagazzinati 500GB di dati giornalieri per un'azienda di medie dimensioni (con una crescita annuali di circa il 75%), con un fattore 1:1 tra server e la replica di backup.

[8] https://www.sollievoit.com/much-bandwidth-snmp-monitoring-consume/

[9] https://www.chaossearch.io/hubfs/C2020/White%20Pages/ChaosSearch%20ELK%20Stack%20Total%20Cost%20of%20Ownership.pdf?hsLang=en

4 Analisi dei casi applicativi

In questa sezione vengono scelti ed analizzati i casi d'uso di riferimento per la realizzazione di questa attività di ricerca. Nello specifico, vengono raccolti i requisiti esposti nella fase di introduzione (Sezione 2) per determinare alcuni possibili scenari reali a cui il software di controllo deve essere in grado di reagire. Questa fase è propedeutica quindi per uno sviluppo corretto del software in merito ai requisiti definiti.

Sono stati definiti due possibili use case ma ciò non vieta di aggiungerne ulteriori in futuro nell'eventualità che emergano fuori nuovi eventi non considerati. Nella specifica vengono inclusi gli attori principali, le condizioni che avviano il caso d'uso e le azioni svolte in sequenza dagli attori coinvolti.

4.1 Caso applicativo #1: Generazione di traffico malevolo

Il primo scenario analizzato è il più elementare che può gestire l'unità logica di controllo (l'SDN controller). Questo evento fa riferimento ad un aspetto di sicurezza in cui un componente del sistema, per la presenza di bug nel codice o per altre vulnerabilità al suo interno sfruttate da utenti maliziosi per cyber-attacchi, inizia ad immettere del traffico malevolo non in linea con il flusso usuale o provocare un attacco DoS e necessita di essere isolato dalla rete per garantire la normale continuità di funzionamento dell'intera struttura e la sua integrità [Pillitteri2014].

Per semplicità si prende come attore principale un dispositivo della rete elettrica, ad esempio, l'IED (Intelligent Electronic Device) ma questo comportamento si può estendere a tutti i nodi monitorati. Gli IED sono dispositivi messi in campo massivamente nei sistemi automatici di erogazione di energia grazie alle sue proprietà di integrazione ed interoperabilità. Essi sono dei box con all'interno principalmente interruttori di controllo e relay che permettono di monitorare il circuito interno e quelli esterni, accedere ai dati delle stazioni, funzionalità PLC e molto altro. Sono costruiti con capacità di computazione e di trasmissione/ricezione dei dati in digitale e possono quindi essere collocati in una rete LAN e comunicare con gli altri dispositivi.

L'IED infetto trasmette in rete i suoi dati, che circolano insieme ai pacchetti immessi dai nodi sani i quali confluiscono entrambi nella base dati comune per essere indicizzati (Elasticsearch). Qui svolge un ruolo determinare l'applicazione adibita ad analizzare in tempo reale le informazioni raccolte, che deve stabilire in breve tempo quali siano i dati corrotti e capire chi sia la sorgente di tale traffico. Soltanto individuando il responsabile che sta generato dati non consoni, essa è poi capace di interagire con l'unità logica di controllo per dare i giusti comandi operativi. Il controller SDN deve poter agire in maniera non ambigua, individuando in quale area della rete si deve intervenire e come intervenire (in questo scenario bloccando l'interfaccia a cui è collegato il dispositivo target). Bisogna quindi stabilire come avviene la comunicazione e lo scambio di informazioni tra applicazione e controller, per consegnare all'ultimo i giusti parametri e le azioni da eseguire per risolvere il problema. A questo punto l'SDN controller avendo quanto necessario per l'intervento, andrà ad iniettare delle configurazioni sui router/switch per isolare l'IED dal resto della rete.

4.2 Caso applicativo #2: Failure di un componente

Il secondo use case analizzato riguarda la capacità del sistema di adattarsi al verificarsi di eventuali guasti, malfunzionamenti e perdita di connettività di uno o più elementi dell'infrastruttura. L'attore che si presta maggiormente ad essere preso ad esempio per questo scenario, e da cui derivano le maggiori criticità emerse da una sua assenza è il nodo "fog", ossia un nodo "concentratore" di dati ed elaborazione che si pone in periferia della rete, in prossimità dei dispositivi fisici (IED). Questo nodo evita di dover trasferire immediatamente tutti i dati ad un elaboratore presente in cloud, nonché è in grado di ospitare sistemi di retroazione locali, che permettono di prendere decisioni intelligenti anche a fronte della mancanza di connettività verso i datacenter remoti.

Infatti, nelle Smart Grid negli ultimi anni si è passato da un approccio centralizzato, dove i dati vengono convogliati in un sistema centrale per essere elaborati e memorizzati (il Cloud), ad un approccio distribuito, in cui i tanti nodi che offrono potenza di calcolo e di archiviazione si presentano ai margini del campo a ridosso dei dispositivi di sensoristica, permettendo notevoli vantaggi. Questa seconda soluzione viene chiamata Fog/Edge Computing e il suo elemento principale è appunto il nodo fog. Le due soluzioni comunque molto spesso coesistono in un ecosistema basato su Cloud, in quanto il nodo fog ricopre lo strato intermedio che permette la comunicazione tra gli elementi periferici della rete e le risorse in Cloud se necessario utilizzarle.

I flussi dati generati dai dispositivi della rete elettrica verranno immessi in un nodo fog primario fino al corretto funzionamento di questo. Nel momento in cui si dovesse verificare una qualche anomalia, l'applicazione di analisi (o direttamente l'SDN controller) deve redirigere il traffico in un secondo nodo fog (il quale già dovrebbe contenere i dati del nodo fog primario, per fornire ridondanza e per non dover spostare in batch tutti i dati di backup). Per far questo, come per il caso d'uso #1, alcune configurazioni devono essere iniettate sul dispositivo da parte del controller, indicando il percorso alternativo che dovrà d'ora in avanti seguire il traffico, fino al ripristino del nodo fog primario.

5 Proposta architetturale

Facendo riferimento ai capitoli 3 e 4, è ora possibile ideare una vista topologica che raggruppa quanto descritto nelle sezioni precedenti, ovvero elencare i componenti che operano nella rete elettrica e dove sono ubicati, stabilire i flussi comunicativi tra di essi (distinguendo tra quelli di rete e di processo) e le scelte tecnologiche effettuate.

5.1 SDN controller

Per quanto concerne l'opzione SDN controller preferenziale è bene effettuare un'analisi valutativa in profondità delle 3 unità logiche di controllo (Ryu, OpenDaylight e ONOS). Ogni piattaforma ha il suo design e differenti casi d'uso di impiego dipendono non solo dalle features offerte ma anche dall'organizzazione che ha messo nel mercato il progetto. Le proprietà prese in considerazione per l'analisi dei controller sono:

- Architettura interna e resilienza
- Modularità
- Scalabilità
- Interfacce
- Community

Per l'analisi comparativa dell'architettura, ci sono dei trade-off da considerare quando si confrontano delle alternative SDN controller centralizzate con quelle decentralizzate e scalabili. ONOS e OpenDaylight implementano il clustering nativamente e forniscono fault tolerance grazie alla presenza di un numero dispari di istanze del controller. Nell'eventualità in cui sul nodo master si verifica un guasto, un nuovo leader viene selezionato per prendere il controllo della rete. La differenza tra ONOS e ODL sta nel processo di scelta del nuovo leader: mentre il primo si focalizza di più a fornire la proprietà cosiddetta "eventual consistency"[10], il secondo si impegna a garantire l'alta disponibilità. Ryu invece non ha nessun meccanismo di clusterig e deve appoggiarsi a tool esterni per mantenere la disponibilità. Questo semplifica di molto l'architettura del controller ed elimina l'overhead dovuto al mantenere le informazioni di stato distribuite, ma l'alta disponibilità del controller si raggiunge solamente avendo più istanze con configurazioni identiche.

Per la modularità e l'estensione, ONOS ed ODL hanno dei meccanismi built-in per caricare moduli a run-time, al costo ovviamente di aumentare il carico computazionale di ognuno. Ryu ha una semplice infrastruttura e gli utenti che lo vogliono utilizzare secondo i loro bisogni devono scrivere del codice aggiuntivo. Da una parte questo richiede del tempo maggiore prima del deploy, nonchè programmatori sufficientemente esperti, ma dall'altra parte si ha piena flessibilità e margine di azione sul controller.

In termini di scalabilità, come già enunciato, solamente ONOS e ODL offrono delle funzionalità per mantenere un cluster. Entrambe le piattaforme hanno un datastore distribuito che condivide lo stato e permette ai controller di adattarsi automaticamente nel caso di una partizione del cluster. Quando la dimensione della rete aumenta, diventa impraticabile per un singolo nodo gestire il carico computazionale derivante dalle richieste degli switch/router. Distribuendo allora i controller geograficamente e partizionando la rete in aree più piccole, diminuisce la necessità di avere un singolo cluster massivamente scalabile.

Considerando i requisiti di compatibilità per il controllo southbound, ONOS, ODL e Ryu includono protocolli che vanno oltre il semplice Openflow. Il punto chiave determinante è svolto però dalle API northbound

[10] Questa proprietà garantisce che esiste un tempo t* nel futuro (non si sa quando) in cui tutte le istanze saranno allineate con le stesse conoscenze. Solitamente, nei db distribuiti si sacrifica la Consistency (che diventa "eventual") per avere Availability e Partition Tolerance, dal momento che per il CAP Theorem non si possono avere tutte e tre.

offerte: ONOS e ODL offrono la più ampia fetta di interfacce con gRPC e RESTful API e quindi più facili da integrare, mentre Ryu ha un'offerta limitata di interfacce REST rispetto ai primi due.

Infine, la community. Sia ONOS che ODL fanno parte della stessa community di sviluppatori/utenti, la Linux Foundation Networking. Molti player internazionali sono coinvolti nello sviluppo e manutenzione di questi progetti, fattore che ne aumenta la longevità e la sicurezza nel lungo periodo. Ryu è ben supportato e data la natura emergente del campo SDN, potrebbe avere un futuro successo. Rimane però tutt'ora una soluzione semplice ed adatta ad un primo testing.

Dall'analisi delle proprietà esaminate, Ryu è il controller meno adatto a prestarsi ad un ambiente dinamico e distribuito come quello pianificato. Restano quindi le soluzioni ONOS e ODL. I due controller sono molto simili sia in termini di funzionalità offerte sia di architettura interna, quindi non c'è motivo di decretarne uno migliore dell'altro. La scelta però è caduta su ONOS per la presenza della GUI (Graphical User Interface). Questa è una pagina web che permette di avere una vista grafica del controller (o cluster di controller) e della topologia dei dispositivi indicando qual è per ognuno il nodo master che li gestisce. Ci sono ulteriori viste navigabili tramite il Menù di Navigazione che permette di raggiungere la vista Applicazioni, per leggere l'elenco delle applicazioni installate e quelle disponibili, la vista Cluster Nodes per leggere l'elenco dei nodi nel cluster, la vista Link per elencare tutti i collegamenti nella rete, la vista Host per elencare tutti gli host presenti nella rete, e altre ancora. La GUI inoltre presenta delle scorciatoie che permettono in tempo reale di visualizzare direttamente dalla vista topologica il percorso del traffico che sta fluendo (evidenziandolo in arancione), statistiche come i bits/s o pacchetti/s e nel caso si installino degli intent, mostrare il percorso scelto evidenziando i nodi che verranno attraversati. L'interfaccia della GUI risulta quindi un valore aggiunto non da poco, dato che permette visivamente anche ad un utente non esperto di capire cosa sta succedendo real-time sulla rete e come le decisioni prese dal controller, mappando le regole sui dispositivi, influenzano il percorso del traffico.

5.2 ELK Stack

L'architettura di questo progetto prevede che tutti i dati di monitoraggio finiscano in un datastore distribuito che è appunto Elasticsearch. Guardando però gli elementi necessari per far funzionare le altre tecnologie, ad esempio Netflow, vediamo come ci sia la necessità di un componente come il flow collector per immagazzinare i flow raccolti, ossia un ulteriore server di storage. Quello che si vorrebbe è integrare tutti i componenti di storage in un unico datastore (Elasticsearch) in cui convogliano i dati delle varie tecnologie presenti. Il tool Logstash mette a disposizione dei moduli che servono per ingerire i dati e visualizzarli mediante dashboard costruite ad hoc. Esso ne ha uno per Netflow (Logstash Netflow Module) che semplifica la collezione, la normalizzazione e la visualizzazione dei flow. Questo modulo però risulta deprecato dalla versione 7.4.0 di ELK e rimpiazzato con il modulo di Netflow di Filebeat che supporta le versioni 1, 5, 6, 7, 8 e 9, così come IPFIX. I record Netflow esportati sono arricchiti dal modulo con dei campi tipici dei flow di tipo Netflow e IPFIX e si utilizza Elasticsearch Ingest Node per arricchire i flow record con informazioni di geolocalizzazione sugli endpoint IP.[11]

L'ingest node viene utilizzato per effettuare delle pre-elaborazioni su un documento prima che questo venga indicizzato o ritornato all'utente. Per effettuare una pre-elaborazione del documento è necessario definire una pipeline che specifica una serie di processors, ed ogni processor modifica il documento in un qualche modo. Esso è abilitato di default e introdotto per eliminare l'utilizzo di Logstash in quei casi in cui il filtraggio e l'arricchimento dei dati risultano dei task semplici (disporre di un server aggiuntivo per Logstash risulta una soluzione non efficiente). Il processor che permette di aggiungere queste informazioni è il GeoIP processor

[11] https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-module-netflow.html

presentato nella Sezione 3.3.4. Il componente che si presta maggiormente ad arricchire i dati con queste informazioni è il router, prima di esportarli ad Elasticsearch, o su un server. Per SNMP, non c'è la possibilità di sfruttare dei Beat per effettuare le richieste specifiche del protocollo. Il plugin SNMP offerto da ELK è per Logstash, che permette di effettuare il poll verso dispositivi di rete per ottenere le informazioni relative al loro stato corrente. Questo plugin include già le MIB IETF e quindi non necessitano di essere importate, mentre altre MIB devono essere importate manualmente.

5.3 Architettura logica

Facendo riferimento allo use case #2 esposto in sezione 4, per fornire fault tolerance e resilienza è necessario muoversi in un ambiente fortemente distribuito e ridondato, questo grazie alla presenza dei nodi fog. Sembra ragionevole installare quindi sia il controller SDN sia i server ELK in questi nodi, nonché l'applicazione di analisi dei dati monitorati. Infatti, alcune proprietà intrinseche del nodo fog sono l'interazione in tempo reale, interoperabilità e federazione (i componenti devono essere in grado di interagire fra loro e i servizi devono essere federati tra domini differenti) e bassa latenza (nell'elaborazione dei dati provenienti dai dispositivi della rete elettrica e nella reattività di controllo). Disponendo quindi questi elementi all'interno dei nodi fog si possono sfruttare queste proprietà. Per Elasticsearch, si può utilizzare la funzionalità di CCR (Cross Cluster Replication, Sezione 10.2.3) per replicare ed allineare i dati del cluster.

La vista topologica, in definitiva, potrebbe avere un aspetto di questo tipo:

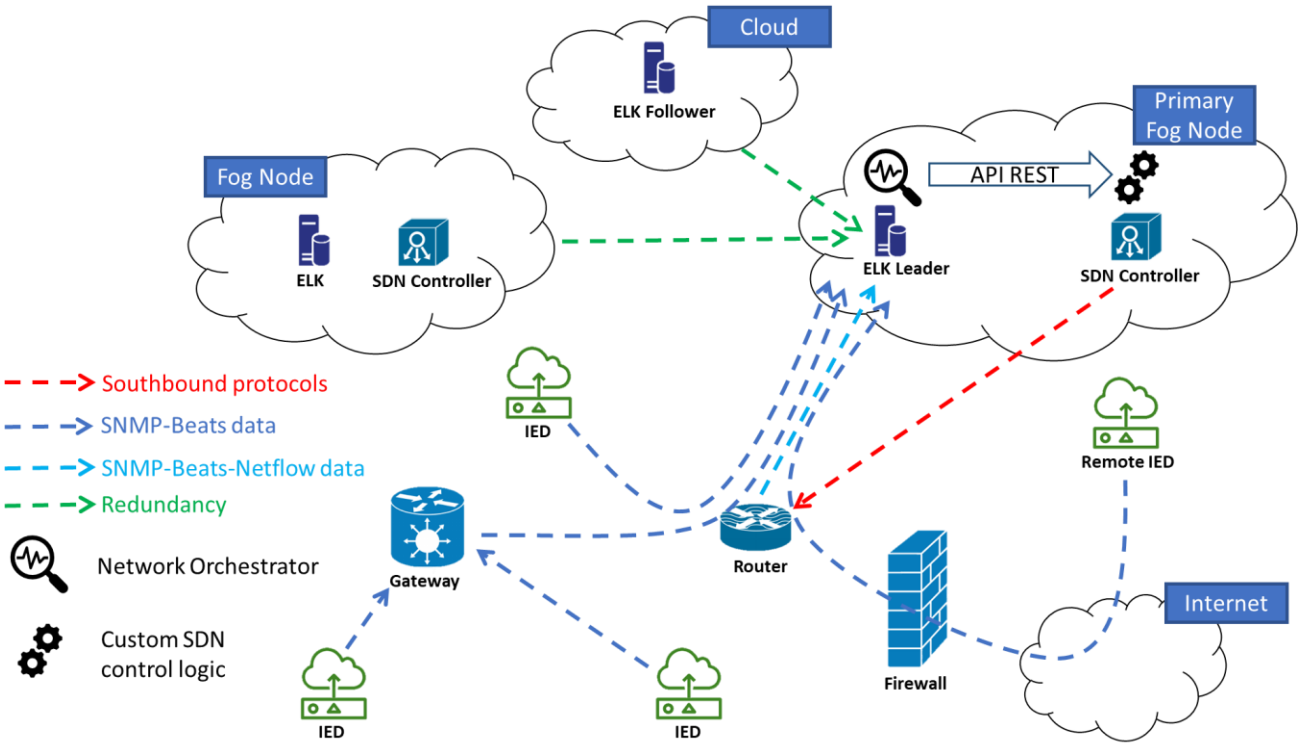


Figure 1 - Network/control data

Nella figura 1 vengono evidenziati con colori differenti le interazioni relative ai flussi di rete, con i protocolli utilizzati e i dati trasmessi tra le varie entità. Al router di campo arrivano le metriche di monitoraggio prodotte dai Beat e dal protocollo SNMP (freccia tratteggiata in blu), installati come sonde negli IED locali e remoti, che, essendo normali pacchetti IP, vengono inoltrati al componente incaricato di creare il "data lake" (ELK), necessario al componente di orchestrazione per poter prendere le proprie decisioni. Inoltre, il router riveste

anche il ruolo di flow exporter, esportando i flussi NetFlow verso il flow collector Elasticsearch presente sul nodo fog primario (freccia tratteggiata in azzurro), rappresentato con una nuvola in alto a destra. Sul nodo fog, inoltre, sono in esecuzione il Network Orchestrator (NetOrch), che include un'applicazione di Machine Learning (ML) in grado di analizzare tutti i dati di rete e di monitoraggio per rilevare violazioni di sicurezza e un'istanza dell'SDN controller ONOS, come esplicato nel caso d'uso #1.

Il NetOrch ha anche il compito di intraprendere eventuali contromisure in presenza di eventi inattesi (es. guasto dell'infrastruttura, richiesta di nuovi servizi a qualità garantita, attacchi di sicurezza ad uno o più nodi). In questo caso si interfaccia con il controller SDN mediante API northbound, tipicamente richiamando delle API REST per attivare la logica di retroazione sulla rete (nel disegno rappresentato dal singolo router a cui vengono iniettate le nuove regole di forwarding). La comunicazione tra il router e il controller è mediata da protocolli southbound (freccia tratteggiata in rosso) quali OF e Netconf. Infine, la ridondanza tra i dati (freccia tratteggiata in verde) si ha disponendo in un nodo fog aggiuntivo una seconda istanza di ONOS e di Elasticsearch i quali ricevono gli stessi dati della loro controparte primaria, per intervenire tempestivamente con tutti i dati necessari per rimpiazzarli nell'eventualità ce ne fosse bisogno. Nel cloud, seguendo il modello della CCR, si ha un'istanza di Elasticsearch per avere un'ulteriore datastore di replica e riveste il ruolo di follower effettuando il pull dei dati dall'istanza leader principale.

Elementi aggiuntivi dell'architettura sono il Gateway SCADA-IP (per gestire l'interoperabilità e la traduzione dei protocolli del mondo SCADA, ancora ampiamente utilizzato in dispositivi elettro-energetici, con quello IP) e un firewall per filtrare il traffico in ingresso proveniente da una rete pubblica (e.g. Internet) e permettere il passaggio dei dati inerenti al monitoraggio degli IED.

La figura seguente (Figura 2) evidenzia invece il flusso del traffico generato dai processi/applicazioni eseguiti nei dispositivi di sensoristica (ad es PMU/PDC). Nell'immagine i componenti e le interazioni presenti sono relativi a MQTT, protocollo di trasporto dei messaggi basato sul paradigma publish/subscribe, utilizzato negli ambienti in cui la banda di rete è limitata (overhead dei pacchetti minimale) e in quanto esso richiede pochissime risorse hardware. Il pattern publish/subscribe (pub/sub) fornisce un'alternativa al modello client/server in cui il client comunica direttamente con gli endpoint. Esso infatti separa il client che manda il messaggio (il publisher) dal client (o dai client) che riceve il messaggio (il subscriber). Publisher e subscriber non vengono mai a contatto direttamente tra loro e non sanno neanche della loro esistenza reciproca. La loro connessione è infatti gestita da un terzo componente, il broker. Il compito del broker è quello di filtrare i messaggi in base al subject e distribuirli ai subscriber correttamente. Ogni messaggio infatti contiene un topic (o subject) che viene utilizzato dal broker per determinare i client interessati a ricevere il messaggio. Per ricevere un messaggio di un topic di interesse, i client mandano un messaggio di Subscribe al broker MQTT, contenente la lista delle iscrizioni e il topic relativo di ciascuna di esse. Nella foto riportata in alto, gli IED svolgono il ruolo di publisher MQTT, nel nodo fog si ha un server che svolge il ruolo di MQTT broker e dei server di storage con il ruolo di subscriber MQTT.

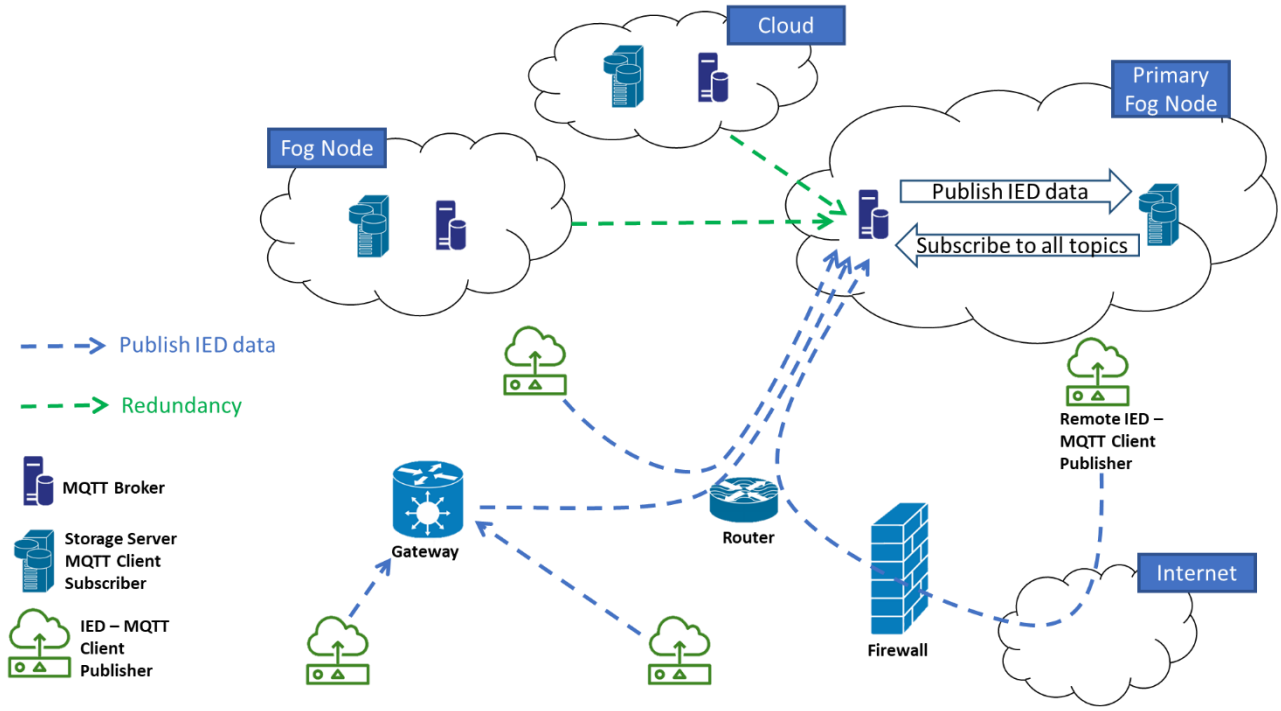


Figure 2 - Data process

6 Implementazione

L'obiettivo di questo progetto è riuscire a controllare un'intera rete reale mediante SDN e tecnologie complementari di monitoraggio. Tuttavia, uno dei problemi riscontrati risulta come stabilire la comunicazione tra il controller e i dispositivi di rete i quali utilizzano protocolli non compatibili con quelli di un controller open-source. Contattando la community di ONOS infatti, si è venuti a conoscenza che i protocolli southbound veramente supportati sono Netconf e OpenFlow. Tuttavia, per quanto riguarda il primo non si riescono a trovare al momento soluzioni open-source virtualizzate di dispositivi che permettono di essere configurati mediante tale protocollo, mentre il secondo non è comunemente supportato dai dispositivi fisici reali, mentre è ampiamente diffuso tra i dispositivi virtualizzati.

Per fornire quindi una dimostrazione quanto più prossima alla realtà utilizzando le soluzioni software disponibili e compatibili con il controller scelto, si è suddivisa la parte implementativa in due fasi. In una prima fase si è deciso di stabilire una rete virtuale emulata basata sul protocollo OF; per semplicità lo strumento più adatto a questo scenario è *mininet*. In una seconda fase il software di virtualizzazione di rete mininet verrà sostituito con switch OF ma operanti in container, creando pertanto una rete emulata che nell'insieme è composta da nodi reali (host Linux) che possono essere connessi ai nodi di rete emulati su container. È da notare infine che, sebbene OF non sia così diffuso su apparati fisici, è possibile utilizzare hardware special-purpose a basso costo con il supporto a Linux (ad es. BananaPI) che creare degli apparati di rete compatibili OpenFlow.

6.1 Mininet

Mininet [Mininet] è un emulatore di rete che permette di eseguire un insieme di host, switch, router e link in un singolo kernel Linux. Esso utilizza la virtualizzazione per far sì che un singolo sistema possa esser visto come un una rete completa. Un host in ambiente mininet si comporta come una macchina reale, ovvero ad esempio è possibile effettuare SSH (Secure Shell) verso di esso ed eseguire programmi user (inclusi quelli installati nel sistema Linux sottostante). I programmi possono inviare pacchetti attraverso quelle che sembrano vere interfacce Ethernet, con una precisa velocità di trasmissione e di ritardo. I pacchetti vengono processati da quelli che danno l'impressione di essere switch e router reali, con buffer di accodamento dei pacchetti con una certa lunghezza.

Anche se le prestazioni (ad es. in termini di throughput TCP) ottenute da una rete mininet non sono necessariamente coincidenti con quelle di una rete reale, dal punto di vista funzionale le due soluzioni (fisica o emulata con mininet) possono essere considerate equivalenti. In breve, gli host, gli switch, i link e i controller virtuali sono oggetti reali, sono soltanto creati mediante software invece di avere una struttura hardware e quindi il loro comportamento è simile. Infatti, è possibile creare una rete mininet che simula una rete hardware, e viceversa una rete hardware che simula una rete mininet, ed eseguire le stesse applicazioni o codice binario sulle due piattaforme.

Vantaggi di mininet:

- Configurare una nuova rete richiede pochi secondi.
- Permette la costruzione di topologie personalizzate che vanno dal singolo switch, a topologie più grandi come Internet, un datacenter o qualsiasi altra rete.
- Permette di cambiare il forwarding dei pacchetti, in quando gli switch mininet utilizzano il protocollo OF.
- Permette di creare reti complesse scrivendo script in Python.

- Mininet è un progetto open-source, permettendo quindi di esaminare il codice sorgente e modificarlo.

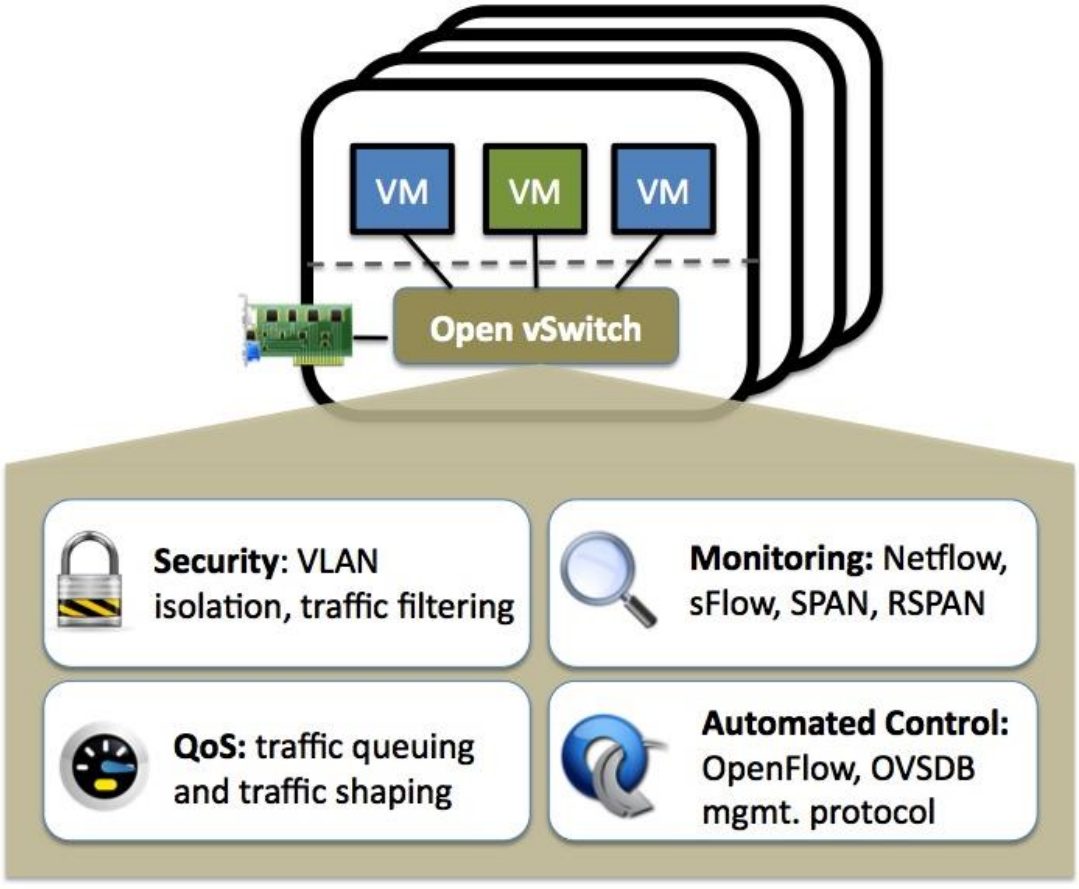
6.2 Open vSwitch

Open vSwitch, abbreviato in OVS, è uno switch software multilayer rilasciato sotto la licenza open-source Apache 2. Il suo scopo è quello di supportare le interfacce standard di management e permettere di programmare e controllare le funzioni di forwarding. OVS è ben predisposto ad operare come switch virtuale in ambienti come le Virtual Machine (VM). Oltre ad esporre un controllo standard e la visibilità delle interfacce dei livelli di rete virtuali, è stato progettato per supportare la distribuzione su più server fisici. OVS supporta le tecnologie di virtualizzazione basate sul sistema operativo Linux, incluse Xen/XenServer, KVM e VirtualBox. Il codice è scritto in una piattaforma C ed è facilmente portabile in altri ambienti.

La versione più recente di OVS ha le seguenti proprietà:

- Lo standard 802.1Q per il supporto alle virtual LAN (VLAN) con porte in modalità access e trunk.
- NIC (Network Interface Card) bonding con o senza LACP (Link Aggregation Control Protocol).
- NetFlow e sFlow per il monitoraggio.
- Configurazione della QoS e delle policy.
- OpenFlow versione 1.0 e numeroso estensioni.
- Alte prestazioni di forwarding usando un modulo kernel Linux.

Il modulo kernel Linux incluso è supportato dalla versione 3.10 di Linux.



Vediamo ora i principali componenti dello switch:

- ovs-vswitchd, il demone che implementa lo switch con il modulo kernel Linux per uno switching basato su flow.
- ovsdb-server, un database leggero che il demone ovs-vswitchd usa per ottenere a sua configurazione.
- ovs-dpctl, un tool per configurare il modulo kernel dello switch.
- ovs-vsctl per interrogare ed aggiornare le configurazioni del demone ovs-vswitchd.
- ovs-appctl per inviare comandi al demone OVS.

Open vSwitch inoltre fornisce i tool:

- ovs-ofctl per interrogare e controllare gli switch OF e i controller SDN.
- ovs-pki per creare e gestire le chiavi pubbliche degli switch OF.
- ovs-testcontroller, un semplice controller SDN OpenFlow che può essere utilizzato in fase di test.
- Una patch di tcpdump che permette di analizzare i messaggi OF.

Negli hypervisor di Linux si utilizza lo switch L2 built-in del kernel, chiamato Linux-bridge, per inoltrare il traffico tra le varie VM e con il mondo esterno. Perché quindi si introduce OVS? La risposta è che OVS è progettato per l'impiego in ambienti multi-server virtualizzati per cui il precedente Linux-bridge non è ben ottimizzato. Questi ambienti sono caratterizzati da endpoint altamente dinamici con astrazioni logiche che spesso prevedono l'integrazione di dispositivi hardware special-purpose per la computazione offloading. Esso quindi si focalizza nell'automazione e nel controllo di reti dinamiche in larga scala basate su ambienti Linux virtualizzati. [12]

6.3 CORE

CORE (Common Open Research Emulator) è uno strumento di emulazione per creare reti virtuali. Essendo quindi un emulatore si costruisce una rappresentazione di una rete reale che viene eseguita in real-time. La rete virtuale emulata può essere connessa a delle reti o dispositivi fisici e fornisce un ambiente per eseguire applicazioni reali e protocolli portandosi dietro i vantaggi dei tool del sistema operativo Linux.

L'uso tipico di CORE è per la ricerca in ambito di rete, per dimostrazioni, test di applicazioni e piattaforme, valutazione di scenari di rete, studi di sicurezza e per aumentare la dimensione della rete fisica. Le principali caratteristiche di CORE sono l'efficienza e la scalabilità, l'esecuzione di applicazioni e protocolli senza apportare modifiche e la presenza di una GUI per il drag-and-drop di nodi di rete. CORE è rilasciato alla comunità open-source sotto la licenza BSD.

Il framework su cui è basato CORE viene eseguito in Linux e usa il suo namespace per creare i nodi di rete.

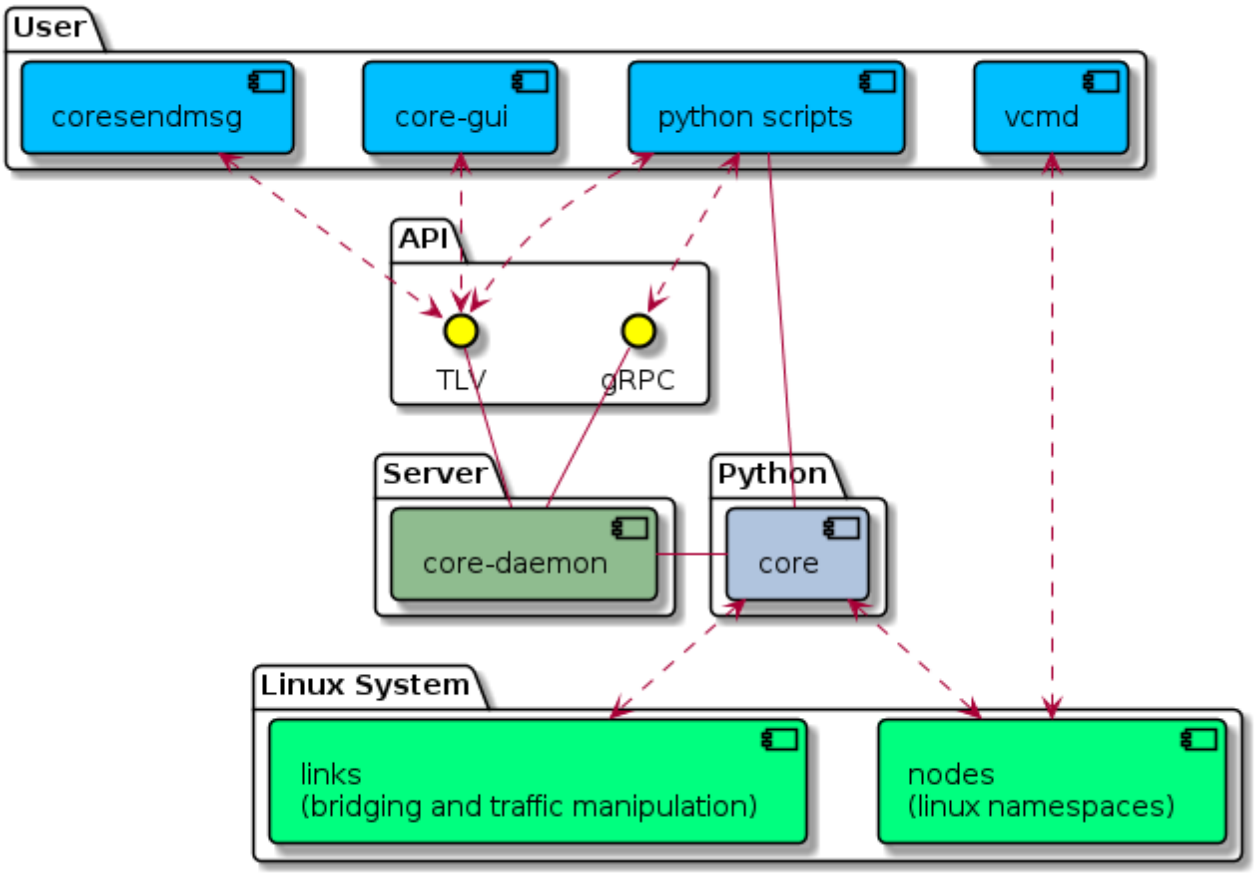
I namespace di rete di Linux, chiamati anche netns, sono la tecnica di emulazione principale utilizzata da CORE. Essi sono disponibili nelle recenti distribuzioni di Linux come feature di default. Ogni namespace ha il suo ambiente di processo e lo stack di rete privato ma tutti condividono lo stesso filesystem in CORE.

Questi nodi sono collegati tra loro usando Linux-bridge e interfacce virtuali. Le sessioni CORE sono un insieme di nodi e link che operano per uno specifico scopo. L'emulatore può creare ed eseguire più sessioni alla volta.

Per quanto concerne l'architettura, i componenti principali sono:

[12] https://www.openvswitch.org/

- core-daemon: demone che gestisce tutte le sessioni, i nodi ed i link per una data rete. Il demone è controllato attraverso l'interfaccia grafica. Per la creazioni dei nodi si utilizzano programmi in Python.
- core-gui: permette la creazioni di nodi e di link tramite GUI e comunica con il core-daemon via TLV (Type-Length-Value) API. Durante l'esecuzione dell'emulazione aprire i terminali dei vari nodi emulati.
- coresendmsg: command line per inviare messaggi TVL API al demone core.
- vcmd: command line per inviare comandi shell ai nodi.



CORE usa il concetto di servizi per specificare quali processi o script eseguire in un nodo quando esso viene eseguito. I nodi di livello 3 come i router e i PC sono definiti dai servizi che eseguono. Questi possono essere modificati per ogni nodo e un nuovo servizio può essere creato, così come nuovi tipi di nodo, ognuno dei quali ha un nome, icona e servizi di default differenti. Ogni servizio definisce le directory del nodo, i file di configurazione, i comandi da eseguire all'avvio e allo shutdown e metadati aggiuntivi. Tra i servizi già disponibili (BGP, OSPFv2, OSPFv3, RIP, Firewall, IPsec, NAT, VPN Client, VPN Server, ecc…) sono inclusi alcuni relativi al mondo SDN, tra cui OVS.

Per iniziare con l'emulazione, bisogna eseguire la GUI, digitando da CLI il comando core-gui (l'interfaccia grafica può essere eseguita come utente normale su Linux) e il demone core (che deve essere eseguito come root). Una volta aperta l'interfaccia grafica (modalità Edit), è possibile trascinare nel foglio bianco che appare

i nodi presenti alla sinistra nella toolbar e questi possono essere configurati effettuando un doppio click su di essi. Una volta che la configurazione dei nodi è completata, premendo il bottone Start in verde si istanzia la topologia all'interno del kernel Linux (modalità Execute). Da questo momento in poi, l'utente può interagire con le macchine emulate in esecuzione effettuando un doppio click su ciascuna di esse. Durante l'esecuzione la toolbar di editing scompare, rimpiazzata da un'altra che comprende i tool da poter utilizzare durante la fase di Execute. Premendo il bottone Stop in rosso, l'emulazione viene distrutta e si ritorna in modalità Edit.

La toolbar iniziale della modalità di Edit è una riga di bottoni raggruppati in sottomenù, i più utilizzati ed interessanti sono:

- Router, con i servizi di routing per l'inoltro dei pacchetti.
- Host, emula una macchina server che esegue un server SSH ed ha una default route.
- PC, macchina base che ha una default route ma non esegue nessun processo.
- Hub, Ethernet hub che inoltra i pacchetti a ogni nodo connesso.
- Switch, Ethernet switch che inoltra i pacchetti ai nodi connessi usando una tabella hash di indirizzi Ethernet.
- RJ45 Physical Interface Tool, per connettere i nodi emulati ad interfacce fisiche reali.

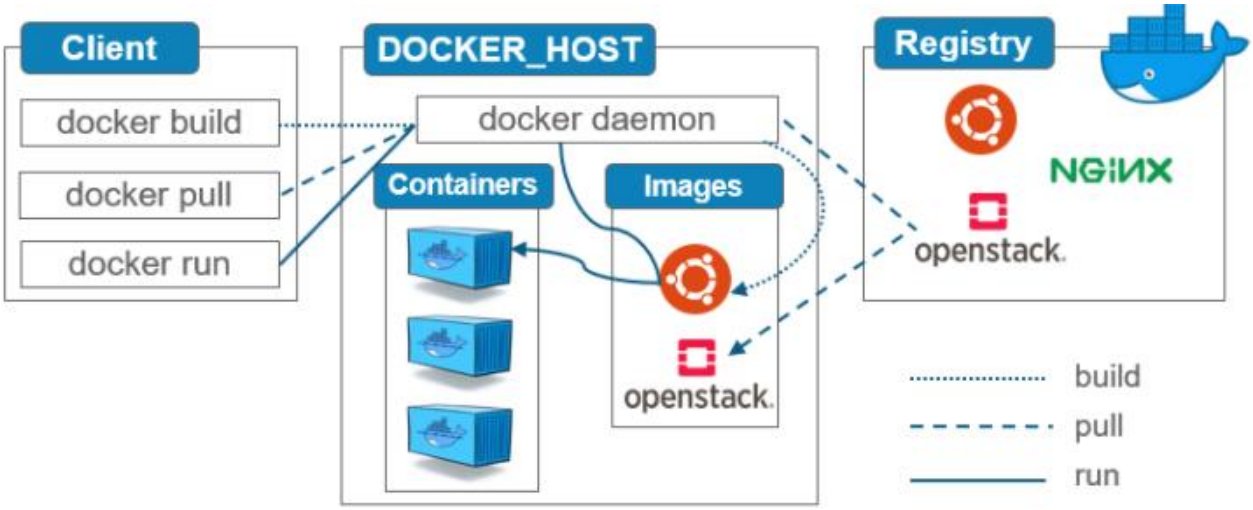
Per la GUI è disponibile una versione BETA denominata Python GUI (core-pygui), che comprende un ulteriore nodo la cui icona è un pallino blu, utilizzato per istanziare container Docker. In fase di configurazione, è possibile specificare l'immagine base, presente nel repository locale, da cui verrà istanziato il container in fase di Execute [CORE].

6.4 Docker

Docker è una piattaforma open-source per lo sviluppo, la distribuzione e l'esecuzione di applicazioni. Docker permette di separare le applicazioni dalla tua infrastruttura in modo da velocizzare il deployment del software e di gestire l'infrastruttura nello stesso modo in cui tu gestisci le tue applicazioni. Utilizzando le metodologie di Docker per la distribuzione, per il testing e per la messa in campo del codice si riduce in modo significativo il tempo che intercorre tra la scrittura del codice e l'esecuzione dell'applicazione in ambiente di produzione.

Docker utilizza un'architettura client-server. Il client Docker comunica con il demone Docker utilizzando API REST ed essi possono essere eseguiti sulla stessa macchina host oppure il client si può connettere ad un demone remoto. L'architettura comprende 3 elementi:

- Docker daemon (dockerd): demone che ascolta le richieste provenienti dal Docker client e gestisce gli oggetti Docker quali immagini, container, reti e volumi.
- Docker client (docker): esso rappresenta la principale modalità di interazione utilizzata dagli utenti per interagire con il Docker daemon attraverso le API Docker. Un altro Docker client è Docker Compose. Esso è un tool per definire ed eseguire applicazioni basate su più container Docker. Per configurare i servizi di un'applicazione si utilizza un file YAML (docker-compose.yml) e successivamente, con un singolo comando (docker-compose), si possono creare, eseguire e fermare tutti i servizi definiti nel file di configurazione.
- Docker registries: su un Docker registry sono salvate le immagini Docker. Docker Hub è il registry pubblico che tutti possono usare ed è quello di default ma si possono anche creare registry privati.



La piattaforma permette di eseguire un'applicazione in un ambiente isolato chiamato container. L'isolamento e la sicurezza permettono di eseguire più container in simultanea su un dato host. I container sono leggeri e contengono tutto il necessario per eseguire la tua applicazione, in modo che tu non debba far affidamento a ciò che è attualmente installato sulla macchina host. Docker sfrutta a suo vantaggio alcune delle feature del kernel Linux. In particolare, esso usa il concetto di namespace per fornire l'isolamento ai container: quando si esegue un container, Docker crea una serie di namespace per quel container e l'accesso di ogni suo aspetto è limitato al namespace in cui viene eseguito.

Un container è un'istanza in esecuzione di un'immagine. Esso può essere creato, eseguito, fermato, spostato o eliminato utilizzando le API o la CLI di Docker. Al container si possono connettere una o più reti, può condividere una parte del filesystem con l'host (quello che viene chiamato volume) e si può creare una nuova immagine a partire dal suo stato attuale.Per default, un container è ben isolato da altri container o dalla macchina host ma l'utente può controllare il livello di isolamento.

Il container è quindi definito dalla sua immagine (così come da altre opzioni di configurazione fornite quando lo si crea o lo si esegue). Un'immagine è un template in sola lettura con le istruzioni per la creazione del container Docker. Spesso un'immagine si basa su un'altra immagine a cui sono apportate delle modifiche. Per esempio si può costruire un'immagine basata su un'immagine ubuntu, su cui in aggiunta è installato un web server e la tua applicazione, così come tutti i dettagli di configurazione per eseguirla. Si può creare una propria immagine ma usare anche quelle create da altri e pubblicate in un registry. Per creare una propria immagine, si deve scrivere un Dockerfile con una semplice sintassi per definire i passi per la sua creazione ed esecuzione. Ogni istruzione presente nel Dockerfile crea un nuovo strato nell'immagine. Quando si modifica il Dockerfile e si effettua il rebuild dell'immagine, solo per gli stati che sono stati cambiati si effettua il rebuild: questa è la parte che permette alle immagini di essere leggere, piccole e veloci quando le si comparano ad altre tecnologie di virtualizzazione.

Docker, di per sé, è in grado di gestire i singoli container in modo molto efficace. Tuttavia, l'utilizzo di un numero sempre maggiore di container e applicazioni containerizzate, a loro volta scomposte in centinaia di componenti, l'orchestrazione e la gestione possono diventare complesse, al punto tale da rendere necessario

il raggruppamento dei container per consentire la distribuzione dei servizi (tra cui servizi di rete, di sicurezza e di telemetria) su tutti i container. A questo punto entra in gioco Kubernetes.[13]

6.5 Kubernetes

Kubernetes (chiamato anche K8s) è una piattaforma open-source per la gestione di servizi operanti in container (usando ad esempio Docker) e ne facilita la configurazione e l'automazione. Kubernetes è stato rilasciato alla community open-source da Google nel 2014.

Come descritto nel paragrafo precedente, è una buona prassi utilizzare i container per eseguire le applicazioni. In ambiente di produzione, bisogna gestire tutti i container e nel caso in cui si dovesse verificare un guasto (e.g. crash) ad uno di questi, un altro container al suo posto va eseguito. Questo può essere gestito automaticamente da Kubernetes, che fornisce un framework per eseguire sistemi distribuiti in maniera resiliente. Esso inoltre fornisce:

- Scoperta dei servizi e load balancing. I container in Kubernetes sono esposti utilizzando il nome DNS o il proprio indirizzo IP. Se il traffico che arriva ad un container è troppo elevato, K8s lo distribuisce ad altri container per migliorare la stabilità dell'infrastruttura.
- Orchestrazione dei sistemi di storage. Kubernetes permette di utilizzare automaticamente un sistema di archiviazione di proprio scelta, come datastore locali, provider cloud pubblici, ecc…
- Rollout e rollback automatici.
- Bin packing automatico. Kubernetes utilizza un cluster di nodi per eseguire i container ed è l'utente a definire quante risorse in termini di CPU e RAM ogni container possiede. K8s esegue i container nei nodi messi a disposizione e sfruttare al meglio le loro risorse.
- Self-healing. Kubernetes riesegue i container che crashano, li rimpiazza o termina i container che non rispondono più.

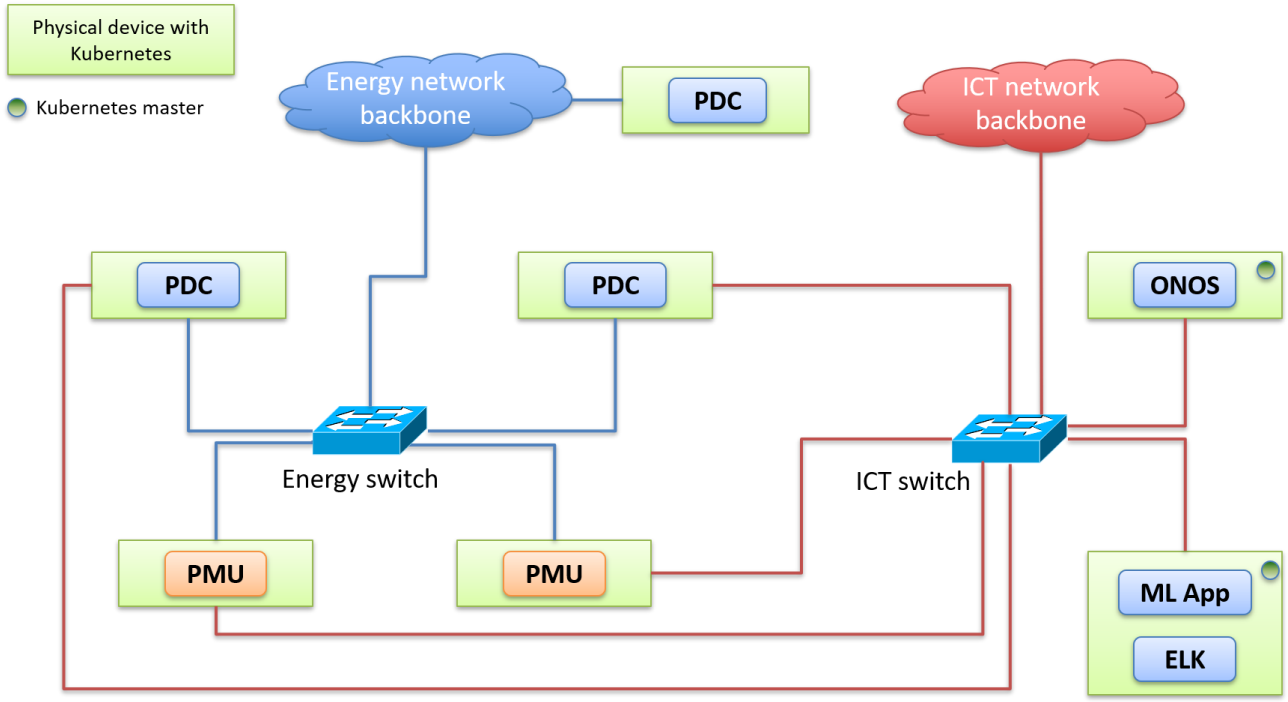
La distribuzione qui utilizzata è K3s, una versione leggera di K8s con un impatto ridotto di circa la metà in termini di risorse di memoria richieste.[14]

[13] https://docs.docker.com/get-started/overview/

[14] https://kubernetes.io/

6.6 Schema topologico

Lo schema riportato in basso mostra la topologia di rete complessiva emulata attraverso l'uso delle tecnologie presentate in questa sezione:



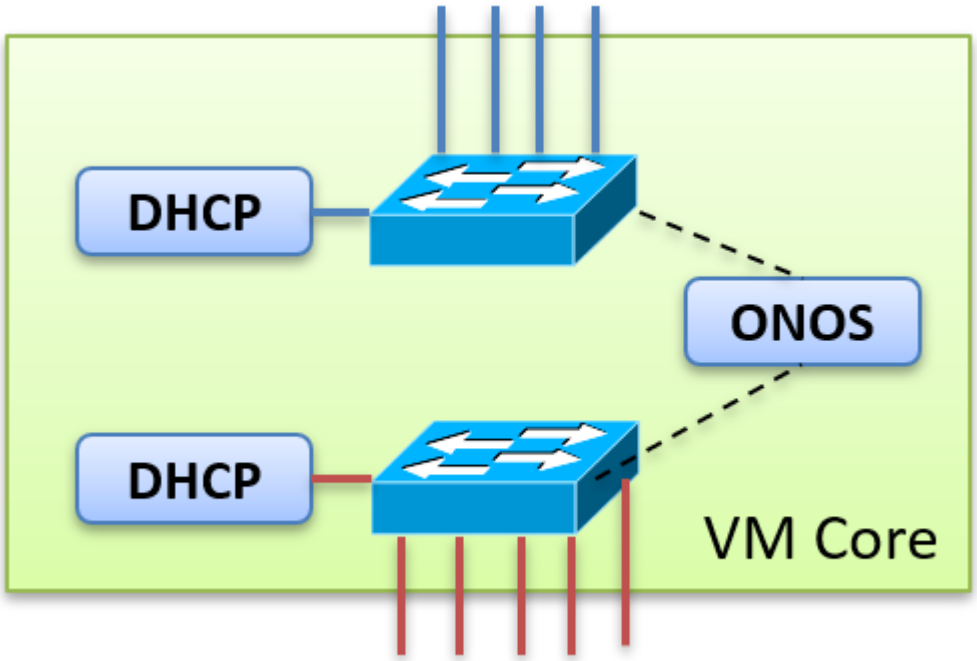
Ogni rettangolo verde rappresenta una VM al cui interno sono eseguiti in container i servizi applicativi quali PMU e PDC (Phasor Data Concentrator) e i servizi atti al monitoraggio (ONOS, Elasticsearch/Kibana, CORE) gestiti da Kubernetes.

Sono presenti due reti L2 per la gestione del traffico:

- La rete Energy (in blu) trasmette il traffico di processo generato tra le varie PDC e PMU. Essa è gestita dallo switch denominato Energy switch.
- La rete ICT (in rosso) trasmette il traffico di monitoraggio, generato dalle sonde (e.g. packetbeat) installate sugli applicativi, e di controllo per la gestione dell'infrastruttura di rete che comprende la comunicazione ONOS-switch per la configurazione delle regole di forwarding e la comunicazione ONOS-ML App per attivare la logica retroattiva del controller SDN. Essa è gestita dallo switch denominato ICT switch.

Per questo motivo, le VM atte ad eseguire le PMU e/o le PDC hanno bisogno di due interfacce di rete, la prima per la connessione verso l'Energy switch per trasmettere i dati di processo, la seconda per la connessione verso l'ICT switch per trasmettere i propri dati di monitoraggio. D'altro canto, per le VM che eseguono ONOS ed Elasticsearch è sufficiente predisporre un'interfaccia di rete verso l'ITC network, in quanto essi non generato nessun dato di processo.

I due switch presenti (Energy switch e ICT switch) sono emulati nella VM di CORE come OVS switch (vedere figura sottostante).



L'ICT switch necessita almeno di due interfacce, una per la connessione verso ONOS e una verso Elasticsearch, in aggiunta un numero di interfacce pari al numero delle VM di processo (PMU/PDC). L'Energy switch deve avere un'interfaccia per ogni VM di processo (PMU/PDC).

In totale quindi, la VM CORE deve avere un numero di interfacce dato dalla somma delle interfacce presenti in ognuno dei due switch.

L'assegnazione degli indirizzi IP può avvenire staticamente per ogni VM, oppure tramite l'impiego di un DHCP per ogni rete LAN.

In alternativa alle due reti fisiche, si potrebbe anche usare una sola rete fisica, utilizzando il concetto di VLAN per definire le due reti logiche (ICT ed Energy). Questo comporta però degli svantaggi: in primo luogo di carattere prestazionale, in quanto la banda per l'inoltro dei dati di processo e dei dati di monitoraggio va condivisa; in secondo luogo per la sicurezza della rete, dato che un attacco su una delle reti può compromettere il funzionamento dell'altra.

7 Prossimi sviluppi e conclusioni

Nelle sezioni precedenti si è evidenziato il possibile uso degli SDN controller e le funzionalità offerte dalle varie opzioni disponibili nel mondo open-source odierno. Questo ha permesso di capire e stabilire l'utilità di questa tecnologia in un sistema distribuito di tipo elettro-energetico che permette, in combinazione ad altre soluzioni per il monitoraggio e di analisi di sistema, di arrivare ad una piena automazione di gestione tramite l'unità logica centralizzata. Il deployment di queste tecnologie permetterebbe di risolvere aspetti di sicurezza e di mitigazione di guasti senza la supervisione e l'intervento umano.

Tra le attività future, si segnalano i seguenti punti.

- Definizione dell'applicazione di analisi dei dati di monitoraggio, e suo sviluppo prototipale, applicato ai casi d'uso definiti nella sezione 4.
- Analisi delle diverse caratteristiche dell'applicazione suddetta quando installata all'interno del controller SDN oppure esternamente ad esso, includendo anche una valutazione prestazionale.
- Definire come scatenare gli eventi che portano al verificarsi dei casi d'uso #1 e #2., ovvero la generazione di traffico malevolo per il primo caso d'uso e il guasto al nodo fog o a uno dei suoi servizi per il secondo.
- Sviluppo della logica retroattiva dell'SDN controller. Oramai si è compreso come ONOS (e in generale tutti i controller open-source) sia soltanto un intermediario tra applicazioni di terze parti e i dispositivi fisici. Questa soluzione rispecchia molto la prima versione di SDN, in cui il controller svolgeva sostanzialmente in lavoro di "passacarte", in quanto un vero controllo automatizzato della rete non è al momento attuabile, se non scrivendo del software aggiuntivo. Sicuramente questo è possibile contestualizzando gli eventi che si vogliono controllare (i casi d'uso #1 e #2) e sfruttando i subsystem e i servizi core messi a disposizione dal controller.

8 Bibliografia

[Elastic] Elastic. (n.d.). *Elastic Stack and Product Documentation*. https://www.elastic.co/guide/index.html?ultron=B-Stack-Trials-EMEA-S-Exact&gambit=Elasticsearch-Documentation&blade=adwords-s&hulk=cpc&Device=c&thor=elastic%20documentation&gclid=Cj0KCQiA9P__BRC0ARIsAEZ6irhF9zCMcFKHZUoUsJixX8Y13c-SIuyH8ato-42okJsWoVMn0kCTy

[ntop] ntop. (n.d.). *nProbe An Extensible NetFlow v5/v9/IPFIX Probe for IPv4/v6*. https://www.ntop.org/products/netflow/nprobe/

[ONOS] ONF. (n.d.). *ONOS*. https://wiki.onosproject.org/display/ONOS/ONOS

[OpenDaylight] OpenDaylight Project. (n.d.). *Platform Overview*. https://www.opendaylight.org/what-we-do/odl-platform-overview

[Pillitteri2014] Pillitteri, V. Y., & Brewer, T. L. (2014, September 25). Guidelines for Smart Grid Cybersecurity (NIST, U.S. Department of Commerce, Ed.). 668. http://dx.doi.org/10.6028/NIST.IR.7628r1

[Rehmani2019] Rehmani, M. H., Jennings, B., Davy, A., & Assi, C. (2019, March). Software Defined Network - Based Smart Grid Communication: A Comprehensive Survey. *IEEE Communications Survey & Tutorials*, 35. 10.1109/COMST.2019.2908266

[RYU] Ryu SDN Framework Community. (n.d.). *RYU SDN Framework*. https://book.ryu-sdn.org/en/html/.

[Mininet] Mininet Team. (n.d.). *Mininet An Instant Virtual Network on your Laptop (or other PC)*, http://mininet.org/

[CORE] Boeing Company. (n.d.). *CORE Documentation*. core Common Open Research Emulator. https://coreemu.github.io/core/

9 Glossario

Casi

API: Application Programming Interface 6

APT: Advanced Package Tool 24

ASN: Autonomous System Number 19

BCC: BPF Compiler Collection 8

BGP: Border Gateway Protocol 13

CCR: Cross Cluster Replication 19

CLI: Command Line Interface 22

CORE: Common Open Research Emulator 39

CPU: Central Processing Unit 5

CRUD: Create Read Update Delete 6

DHCP: Dynamic Host Configuration Protocol 7

DNS: Domain Name System 7

DoS: Denial-of-Service 16

eBPF: extended Berkeley Packet Filter 8

ELK: Elasticsearch Logstash Kibana 5

FIFO: Fist In First Out 12

GB: GigaByte 8

GbE: Gigabit Ethernet 8

gRPC: gRPC Remote Procedure Calls 13

GUI: Graphical User Interface 18

HTTP: Hypertext Transfer Protocol 7

ICMP: Internet Control Message Protocol 7

IED: Intelligent Electronic Device 3

IETF: Internet Engineering Task Force 3

IP: Internet Protocol 3

IPFIX: IP Flow Information Export 3

JSON: JavaScript Object Notation 6

KQL: Kibana Query Language 19

LA: Link Aggregation 12

LACP: Link Aggregation Control Protocol 38

10 Appendice A: Cenni di configurazione

Questa appendice riporta i principali comandi necessari all'installazione e configurazione degli strumenti software utilizzati da questo progetto.

I vari tool sono installati tutti in locale sulla stessa macchina host.

10.1 Setup di ONOS

Il deployment di ONOS è stato effettuato in "Developer mode", ovvero compilando il sorgente di ONOS ed eseguendolo, in modo tale da poter sviluppare i componenti core del controller così come le applicazioni e i driver. Si è svelto Bazel per la compilazione del sorgente, essendo ONOS un progetto ampio composto da molti moduli indipendenti che permettono di essere quindi compilati in parallelo. Per compilare ONOS sono necessari almeno 16 GB di RAM, Sistema Operativo Linux o MAC OS con architettura x86_64 ed è necessario installare i tool Python2 e Python3 (richiesti per il build) e git per effettuare il pull e il push del codice dal repository di ONOS.

Il primo componente da installare è Bazelisk (che è un wrapper di Bazel), tool open-source sviluppato da Google per il supporto alla compilazione. Successivamente si ottiene il codice sorgente utilizzando git

```
git clone https://gerrit.onosproject.org/onos
```

Viene creata una cartella denominata "onos" in cui eseguire il comando bazel per effettuare il build

```
bazel build onos
```

Il quale compila ed assembla il file onos.tar.gz presente nella cartella bazel-bin. Infine per poter eseguire ONOS localmente sulla macchina host di sviluppo lanciare

```
bazel run onos-local -- clean debug
# 'clean' to delete all previous running status
# 'debug' to enable remote debugging
```

Il comando sopra riportato crea un'installazione locale dal file onos.tar.gz ed esegue ONOS in background e inoltre esso mostra a video i log prodotti da ONOS. Per attaccare la CLI (Command Line Interface) eseguire

```
tools/test/bin/onos localhost
```

In questo modo si possono eseguire i vari comandi disponibili nella CLI, come attivare/disattivare le applicazioni. Visitare http://localhost:8181/onos/ui per visualizzare la GUI di ONOS. Molti script e comandi di ONOS necessitano di specificare il path del loro eseguibile per poter essere lanciati. Per far si che essi siano eseguibili in qualsiasi cartella della macchina host, specificare le seguenti due righe[15]

```
export ONOS_ROOT=~/onos
```

[15] https://wiki.onosproject.org/display/ONOS/Developer+Quick+Start

```
source $ONOS_ROOT/tools/dev/bash_profile
```

10.1.1Template Application

ONOS permette attraverso il tool onos-create-app di generare template application, ovvero si appoggia a Maven per generare una struttura base delle applicazioni così come differenti varianti per aggiungere comandi alla CLI ed endpoint REST. Si mostra ora il procedimento per generare lo scheletro di un'applicazione ONOS, che valgono poi per le creazioni di comandi custom CLI e API RESTful ad essa collegati.

L'applicazione creata sarà compilabile e pronta per essere caricata. Per creare la base del progetto bisogna specificare groupId, artifactId e la versione, la cui tripla permette di localizzare l'applicazione univocamente in Maven. Inoltre, si può specificare il nome del package Java in cui il codice sarà generato. Esempio:

```
onos-create-app app org.foo foo-app 1.0-SNAPSHOT org.foo.app
```

Una volta creato lo scheletro, si può compilare e caricare l'applicazione in ONOS. Il progetto sarà generato in una nuova cartella (chiamata nel caso in esame foo-app) e per compilarlo utilizzare maven al suo interno

```
mvn clean install
```

Completato il build, il modulo OSGi e l'archivio sono installati nel repository locale di maven. Per caricare l'applicazione nell'istanza di ONOS, si utilizza il tool onos-app messo a disposizione, che utilizza le API REST built-in per caricare il file .oar al suo interno. Il punto esclamativo che segue il parametro install, permette di attivare l'applicazione immediatamente dopo che l'installazione è avvenuta con successo

```
onos-app localhost install! target/foo-app-1.0-SNAPSHOT.oar
```

A questo punto, dalla console di ONOS, è possibile verificare che l'applicazione sia stata caricata e attivata correttamente (comando apps -s -a). Il codice generato per questa app permette di aggiungere del codice aggiuntivo editando il file src/main/java/org/foo/app/AppComponent.java.

Una volta creata l'applicazione, è possibile attaccarci sopra dei comandi CLI ed endpoint REST custom. I comandi da lanciare sono i medesimi, cambiano i parametri: in onos-create-app sostituire il parametro app con cli o rest rispettivamente

```
onos-create-app cli org.foo foo-app 1.0-SNAPHOST org.foo.app
onos-create-app rest org.foo foo-app 1.0-SNAPHOST org.foo.app
```

E sostituire il parametro install! con reinstall!, dato che l'applicazione è già stata installata

```
onos-app localhost reinstall! target/foo-app-1.0-SNAPHOST.oar
```

Alla fine di questi passaggi verranno creati i file src/main/java/org/foo/app/AppCommand.java, AppWebResource,java e AppWebApplication.java in cui introdurre la logica custom per il comando e l'API REST. Per quest'ultimo, ONOS genera automaticamente la documentazione tramite SwaggerUI per le URI esposte nell'applicazione creata, visitandola all'indirizzo http://localhost:8181/onos/ui/ in cui è possibile richiamare direttamente le funzioni REST.[16]

10.1.2 Network Configuration Service

Il Network Configuration Service (NCF) fornisce supporto per due task:

1. Configurare alcune applicazioni di ONOS.
2. Aggiungere informazioni sui dispositivi, link e le configurazioni dei dispositivi.

Parte delle applicazioni di ONOS richiedono delle configurazioni aggiuntive specifiche oltre a quelle di default, dato che alcune informazioni non possono essere dedotte automaticamente. Il NCF permette appunto di configurarle ed inoltre esso fornisce l'abilità di aggiungere informazioni topologiche alla vista di rete del controller, permettendo di scrivere un programma che legge le informazioni sui dispositivi e sulla topologia, per esempio da un database esterno, e fornirli ad ONOS, senza dover affidarsi a servizi di scoperta automatica. Le configurazioni a questo servizio vengono caricate mediante file JSON.

NCF può essere utilizzato per iniettare informazioni al controller (IP, porta, username, password) relative ai dispositivi di rete per consentire l'interazione utilizzando i protocolli southbound (Netconf, SNMP, REST). Questo permette ad ONOS di sapere quali dispositivi stanno utilizzando Netconf o SNMP come interfaccia comunicativa. Le configurazioni possono essere caricate via API REST o con il comando onos-netcfg.[17]

Entrambe le soluzioni sono state testate, sia per il protocollo Netconf sia per SNMP, utilizzando le configurazioni di esempio presenti nella cartella $ONOS_ROOT/tools/test/configs/. Per Netconf si è utilizzato of-config (wrapper di un'instanza openvswitch, che utilizza il protocollo Netconf e lo traduce in OVSDB in modo da poter comunicare con lo switch) per simulare in software un dispositivo di rete Netconf-aware. Il problema è che questo tool ha delle capacità limitate e non si ha la possibilità quindi di cambiarne il comportamento. Inoltre, si è verificato che gli intenti supportano soltanto OpenFlow come protocollo, in quando andando a specificare degli intenti per dispositivi Netconf o SNMP aware, questi non vengono tradotti e rimangono nello stato di PENDING.

10.2 Setup della pila ELK

Per l'installazione dei tool di ELK Stack, si illustrano i passaggi soltanto per Elasticsearch essendo essi i medesimi per i restanti componenti (tranne le opzioni nei file di configurazione). Quando la si installa, bisogna usare la stessa versione per tutti i tool della pila (in questo caso 7.9.3). I tool sono forniti in package in vari formati e per la distribuzione Ubuntu 20.04.1 LTS utilizzata è disponibile l'archivio nel formato .tar.gz o il package Debian (in questo caso si è scelta la seconda opzione).[18] I componenti di Elastic Stack non sono disponibili nel repository dei package di default, ma possono essere installati con APT dopo aver aggiunto la lista dei sorgenti dei package di Elastic. I prerequisiti necessari per l'installazione comprendono le risorse hardware enunciate nella sezione 2 e la presenza di Java 8.

[16] https://wiki.onosproject.org/display/ONOS/Template+Application+Tutorial

[17] https://wiki.onosproject.org/display/ONOS/The+Network+Configuration+Service

[18] https://www.elastic.co/guide/en/elasticsearch/reference/current/install-elasticsearch.html

Per prima cosa bisogna importare la chiave pubblica con cui sono firmati tutti i package di Elastic:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key
add -
```

Poi aggiungere la lista delle sorgenti nella cartella sources.list.d, dove APT andrà a guardare per nuove sorgenti:

```
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo
tee -a /etc/apt/sources.list.d/elastic-7.x.list
```

Per far sì che APT sia al corrente dei nuovi sorgenti di Elastic, bisogna aggiornare la lista dei package ATP:

```
sudo apt update
```

Infine, installare Elasticsearch con il comando

```
sudo apt install elasticsearch
```

Per avviare Elasticsearch si utilizza systemctl (serve per controllare il manager dei servizi):

```
sudo systemctl start elasticsearch
```

Per abilitare Elasticsearch ad essere eseguito ogni volta che il server si avvia, eseguire il comando:

```
sudo systemctl enable elasticsearch
```

Elasticsearch per default ascolta tutto il traffico proveniente dalla porta 9200 e per eseguire il servizio in locale, aprire il file di configurazione elasticsearch.yml presente nella directory di installazione (/etc/elasticsearch/elasticsearch.yml) con un text editor (e.g. nano) e specificare nel campo network.host il valore localhost:

```
...
network.host: localhost
...
```

Per testare lo stato del servizio si può effettuare una richiesta HTTP oppure utilizzare systemctl:

```
curl -X GET "localhost:9200"
sudo systemctl enable elasticsearch
```

Per installare i componenti restanti della pila ELK i comandi e passi da eseguire sono i medesimi. Kibana è in ascolto per default sulla porta 5601 (per accedere all'interfaccia web visitare quindi localhost:5601) mentre Logstash sulla porta 9600.

Guardiamo ora come vengono configurati i Beat.[19] Filebeat supporta diversi output, ma nel nostro caso i dati da esso generati devono confluire direttamente in Elasticsearch. Nel file di configurazione (/etc/filebeat/filebeat.yml) specificare quindi la sezione output.eleasticsearch nel seguente modo:

```
...
output.elasticsearch:
   # Array of hosts to connect to.
   hosts: ["localhost:9200"]
...
```

Le funzionalità di Filebeat possono essere estese grazie alla presenza dei moduli e quello di nostro interesse è il modulo netflow, che va abilitato lanciando il comando:

```
sudo filebeat modules enable netflow
```

Si può ulteriormente ridefinire il comportamento del modulo di netflow modificando le variabili di configurazione nel file modules.d/netflow.yml. Se non vengono alterate, vengono utilizzate le variabili di default (si ascolta la provenienza di traffico UDP da localhost:2055).

Successivamente va caricato l'index template in Elasticsearch per comunicare come configurare l'indice quando questo viene creato. L'indice viene identificato con un nome che viene utilizzato per indicare l'indice target delle operazioni che lo coinvolgono. Per caricare il template, utilizzare il comando:

```
sudo filebeat setup --template -E output.logstash.enabled=false -E
'output.elasticsearch.hosts=["localhost:9200"]'
```

Inoltre, Filebeat si presenta equipaggiato con numerose dashboard di esempio per Kibana, che permettono di visualizzare i dati di Filebeat in Kibana. Prima di utilizzare la dashboard bisogna caricarla al suo interno:

```
sudo filebeat setup -e -E output.logstash.enabled=false -E
output.elasticsearch.hosts=['localhost:9200'] -E
setup.kibana.host=localhost:5601
```

Il nome di default dell'indice creato è filebeat-* e per verificare che i dati vengono correttamente ricevuti da Elasticsearch si possono utilizzare le API REST oppure utilizzare la web interface di Kibana nella sezione Discover.

Gli stessi passaggi vanno ripetuti per Packetbeat (escludendo il punto in cui si abilita il modulo di netflow).

[19] https://www.elastic.co/guide/en/beats/filebeat/current/configuring-howto-filebeat.html

10.2.1.1 nProbe

Per la generazione di record Netflow si utilizza il tool nProbe. Esso è una sonda software in grado di collezionare, analizzare ed esportare traffico di rete utilizzando il formato standard Cisco Netflow/IPFIX [ntop]. Qui viene utilizzato come elemento che mima il comportamento del router in grado di collezionare ed esportare i flow al collector. In questo caso però nProbe viene configurato per inviare i record a Filebeat, il quale poi grazie al suo modulo permetterà di indicizzarli in Elasticsearch.

Per installarlo digitare nella shell:

```
apt-get install software-properties-common wget
add-apt-repository universe
wget https://packages.ntop.org/apt/20.04/all/apt-ntop.deb
apt install ./apt-ntop-deb
```

Successivamente per attivare la cattura basterà eseguire il comando

```
nprobe -i enp1s0 --collector 127.0.0.1:2055
```

10.2.2 Beat e container

Per tutti i tool della pila ELK c'è la possibilità di effettuare il deployment di questi componenti con container Docker.[20] Questo è stato sperimentato per Packetbeat (in generale quanto si dice vale per tutti i Beat) in quanto si vorrebbe installarli sui dispositivi di sensoristica in modo agile, riducendo il carico computazionale e portandosi dietro tutti i benefici della containerizzazione.

Le immagini Docker dei beats sono disponibili nel registry Elastic Docker (la lista completa è disponibile al seguente link https://www.docker.elastic.com). Per Packetbeat 7.9.2 l'immagine di base da cui è costruito il container è CentOS. Per ottenere un'immagine Docker di Packetbeat basta lanciare il comando docker pull verso il registry Elastic Docker:

```
docker pull docker.elastic.co/beats/packetbeat:7.9.2
```

Le immagini Docker forniscono diversi metodi per configurare Packetbeat. L'approccio convenzionale prevede di fornire un file di configurazione attraverso un volume della macchina host, oppure creare un'immagine personalizzata via Dockerfile con il file di configurazione incluso (si è scelta la prima opzione). Per fornire il file di configurazione packetbeat.docker.yml via volume, bisogna eseguire il comando docker run specificando il punto di montaggio del volume:

```
docker run -d -p 9200:9200 \
  --name=packetbeat \
  --user=packetbeat \
--
volume="$(pwd)/packetbeat.docker.yml:/usr/share/packetbeat/packetbeat.yml:ro"
\
```

[20] https://www.elastic.co/guide/en/beats/packetbeat/master/running-on-docker.html

```
--cap-add="NET_RAW" \
--cap-add="NET_ADMIN" \
--network=host \
docker.elastic.co/beats/packetbeat:7.9.2 \
--strict.perms=false -e \
-E output.elasticsearch.hosts=["localhost:9200"]
```

Per funzionare correttamente, bisogna specificare alcune impostazioni di rete. Packetbeat su Docker viene eseguito come un utente non root, ma per operare liberamente esso richiede alcuni privilegi di rete. Esempio l'opzione NET_ADMIN permette di eseguire varie operazioni di rete (lista completa https://man7.org/linux/man-pages/man7/capabilities.7.html). Di default, il container di Packetbeat ha una rete virtuale isolata, con una visione limitata del traffico. Per connettere il container con la rete della macchina host in modo da vedere il traffico destinato verso la macchina host e da essa generato, basta specificare il comando - -network=host.

10.2.3 Cross Cluster Replication

Come accennato nel paragrafo relativo allo use case #2 (sezione 3.1) e visibile nella proposta architetturale, la ridondanza dei dati in Elasticsearch è di fondamentale importanza per garantire le proprietà di fault-tolerance, disaster recovery e data locality. La pila ELK ci mette a disposizione una feature interessante, che va sotto il nome di Cross Cluster Replication.[21] Questa permette di replicare gli indici tra vari cluster utilizzando un modello attivo-passivo. L'indice che viene ridondato è denominato leader e i documenti al suo interno vengono replicati in uno o più indici read-only, chiamati follower. Prima di aggiungere un indice follower a un cluster, bisogna configurare il cluster remoto che contiene l'indice leader con il supporto alla CCR. Quando l'indice leader riceve delle richieste di scrittura, gli indici follower effettuano il pull delle modifiche avvenute nel leader sul cluster remoto. Il setup della CCR può avvenire in due modalità:

1. Configurazione uni-direzionale, in cui un cluster contiene soltanto indici leader e gli altri cluster contengono solamente indici follower (il cluster contenente indici follower deve eseguire la stessa versione o superiore di Elasticsearch presente nel cluster remoto).
2. Configurazione bi-direzionale, in cui i cluster contengono sia indici leader che follower.

Anche se il setup della CCR avviene a livello di indice, Elasticsearch effettua la replicazione a livello di shard: quando un indice follower viene creato, ogni shard in esso presente richiede le modifiche apportate nell'indice leader al suo corrispettivo shard (implica che il numero di shard presenti nell'indice leader e follower è il medesimo). Elasticsearch deve allora conservare la storia delle operazioni eseguite all'interno di ogni shard dell'indice leader in modo tale che queste possano essere copiate dalle shard degli indici follower e rieseguite. Il meccanismo utilizzato per mantenere la lista delle operazioni va sotto il nome di soft deletes. Una soft delete avviene ogni qualsiasi volta che un documento esistente è eliminato o aggiornato e viene mantenuta in memoria dagli shard leader e messi a disposizione di quelli follower. Queste devono essere abilitate in quegli indici che prenderanno poi il ruolo di leader (abilitate di default dalla versione di Elasticsearch 7.0.0. in poi). L'indice follower creato non può essere utilizzato fino a quando non è pienamente inizializzato. Si avvia il processo di remote recovery in cui viene creata una nuova copia di uno shard nel nodo follower copiando i dati dallo shard leader. Questo è un task che richiede molte risorse di rete (anche se

[21] https://www.elastic.co/guide/en/elasticsearch/reference/current/xpack-ccr.html

questo opera in parallelo) in quanto avviene il trasferimento di tutti i segmenti di file (chunk di 1 MB) dal leader al follower.

10.2.3.1 Installazione

Per impostare la CCR sui cluster, è necessario disporre in entrambi della licenza Platinum (è disponibile una versione gratuita di 30 giorni). Per testarla, è stata installata una seconda istanza, oltre a quella presente in versione Debian, di Elasticsearch e di Kibana (nella versione 7.10.0) dall'archivio in formato tar.gz e configurati per essere eseguiti in porte differenti e in un cluster separato dal primo (ClusterB).

Il setup può essere effettuato sia tramite API REST, sia tramite la dashboard di Kibana.

Per prima cosa bisogna connettere il cluster con l'indice follower (ClusterB) al remote cluster. Andare quindi nella web interface della seconda istanza di Kibana, eseguita sulla porta 5602, nella sezione Dev Tools in cui è possibile lanciare le chiamate REST. Per aggiungere il remote cluster:

```
PUT /_cluster/settings
{
  "persistent" : {
    "cluster" : {
      "remote" : {
        "leader" : {
          "seeds" : [
            "127.0.0.1:9300"
            ]
         }
      }
    }
  }
}
```

Nell'oggetto seeds si specifica l'hostname e la porta utilizzata per la comunicazione tra i nodi nel cluster remoto.

Si può verificare che il cluster si è connesso con successo al cluster remoto lanciando

```
GET /_remote/info
```

Che dovrebbe restituire una richiesta del tipo

```
{
  "leader" : {
      "connected" : true,
      "mode" : "sniff",
```

```
      "seeds" : [
      "127.0.0.1:9300"
      ],
      "num_nodes_connected" : 1,
      "max_connections_per_cluster" : 3,
      "initial_connect_timeout" : "30s",
      "skip_unavailable" : false
  }
}
```

Infine, si crea un indice follower:

```
PUT /ccr_demo_index_replicated/_ccr/follow
{
  "remote_cluster" : "leader",
  "leader_index" : "ccr_demo_index"
}
```

La prima risorsa presente nel path è il nome dell'indice follower che verrà creato (ccr_demo_index_replicated in questo caso) e il corpo della richiesta contiene il nome del cluster (leader) e l'indice leader da replicare (ccr_demo_index).

11 Appendice B: Cenni di configurazione

Questa appendice si differenzia dalla precedente “Appendice A: Cenni di configurazione” in quanto riporta i principali comandi necessari all’installazione e configurazione degli strumenti software in ambiente emulato.

11.1 Setup di Mininet

Installare mininet su Ubuntu è molto semplice. Infatti essa è presente come package nel system repository di Ubuntu.

Eseguire il seguente comando per ottenere le più recenti informazioni sui package:

```
sudo apt-get update -y
```

Per installare mininet e le dipendenze ausiliarie:

```
sudo apt-get install -y mininet
```

Una volta installata si può creare una rete con un singolo comando. Una buona pratica consiste nell’eliminare i bridge esistenti ed i relativi namespace, questo perchè la shell di mininet si può erroneamente chiudere ma i componenti continuano ad esistere. Il comando di cleanup utilizzato è questo:

```
mn -c
```

Mininet da modo di creare reti con diverse topologie. Ad esempio per creare una rete con una topologia con un singolo switch e 4 nodi:

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --
switch=ovs,protocols=OpenFlow13 --topo=single,4
```

dove --controller indica il controller (remote o local) e il suo indirizzo IP; --mac il valore iniziale degli indirizzi MAC; -i la sottorete IP della topologia; --switch il tipo di switch e la versione di OF utilizzata; --topo il tipo di tipologia (linear, minimal, reversed, single, torus ,tree) e i parametri.

11.2 Setup VM

In ogni VM (tranne quella in cui è eseguito CORE) deve essere configurata con il Docker engine per creare i container contenenti le applicazioni ed essere predisposta di un cluster K3s per orchestrarli.

Per installare Docker:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Per la creazione di un cluster K3s, mono-master e mono-node:

```
curl -sfL https://get.k3s.io | INSTALL_K3S_NAME="master" sh -s - server -v 3 -
-write-kubeconfig-mode 644
```

Per una maggiore performance è consigliato disabilitare lo swap:

```
sudo swapoff -a
```

K3s dovrebbe partire automaticamente dopo l'installazione; verificarlo con il comando:

```
k3s check-config
```

Successivamente, per controllare il cluster tramite kubectl (programma CLI per interagire con K3s da console e da terminale) bisogna configurarne l'accesso nel seguente modo:

```
mkdir -p $HOME/.kube
cp -i /etc/rancher/k3s/k3s.yaml $HOME/.kube/config
ls -lar $HOME/.kube
chown $(id -u):$(id -g) $HOME/.kube/config
```

11.3 Setup PMU/PDC

La virtualizzazione dei servizi applicativi PMU/PDC non ricade in questa attività di tesi. Comunque, è stato necessario installare su di essi i tool necessari all'invio dei dati di monitoraggio verso Elasticsearch, in particolare la sonda packetbeat, e il tool per simulare l'attacco di sicurezza, ovvero hping.

Nel Dockerfile della PMU/PDC inserire le seguenti istruzioni:

```
## Added to execute packetbeat
RUN wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | apt-key
add -
RUN apt-get install apt-transport-https
RUN echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | tee
-a /etc/apt/sources.list.d/elastic-7.x.list
RUN apt-get update && apt-get install packetbeat
COPY packetbeat.yml /etc/packetbeat/
COPY packetbeat.yml /usr/share/packetbeat/
COPY packetbeat.yml /usr/share/packetbeat/bin/
#Added to mock PING FLOOD ATTACK with hping
RUN apt-get update -y
RUN apt-get install -y hping3
```

Nel container viene copiato il file packetbeat.yml (presente nella stessa directory del Dockerfile) per configurare packetbeat ad inviare i dati alla VM di Elasticsearch. In breve si devono modificare i campi che indicano l'indirizzo IP (e.g. 192.168.101.7) e la porta in cui Kibana ed Elasticsearch sono in esecuzione:

```
# ================================== Kibana =====================================
setup.kibana:
  host: "192.168.101.7:5601"
```

```
# ================================== Outputs===================================
# Configure what output to use when sending the data collected by the beat.
# ---------------------------- Elasticsearch Output ----------------------------
--
output.elasticsearch:
  hosts: ["192.168.101.7:9200"]
```

Infine, aggiungere all'istruzione CMD il comando per eseguire packetbeat. Ad esempio per la PMU (equivalente per la PDC):

```
CMD service packetbeat start && /tool/iPDC/iPDC-v1.3.1/PMUSimulator-
1.3.1/./PMU
```

11.4 hping

Per simulare l'attacco di sicurezza definito nello use case #1 si utilizza il tool hping. Esso è un tool orientato ad analizzare ed assemblare pacchetti TCP/IP. La sua interfaccia si ispira al comando ping ma hping non è soltanto in grado di inviare ICMP echo request. Esso supporta TCP, UDP, ICMP ed altri protocolli IP, presenta la modalità traceroute e ha l'abilità di inviare file tra canali ed altre feature. hping è usato principalmente come tool di sicurezza ma può essere usato anche per effettuare test sulla rete e sugli host (test sui firewall, port scanning, test sulla rete usando differenti protocolli, ecc…).

In questa attività di ricerca esso viene utilizzato come strumento di attacco per effettuare un DoS, in particolare un Ping flood attack, che un attacco in cui l'attaccante (qui la PMU) invia un enorme quantità di ICMP echo request al dispositivo target (nel nostro caso la PDC). Inondando la PDC con innumerevoli richieste, la rete è obbligata a rispondere con un egual numero di risposte. Ciò fa sì che il servizio target non sia più accessibile e/o disponibile all'uso.

Per utilizzare il comando hping, entrare nella bash del container di una delle pmu e lanciare il comando:

```
sudo hping3 -1 --flood <ip_addr>
```

-1 specifica la modalità ICMP; --flood invia i pacchetti il più velocemente possibile e non mostra le risposte; ip_addr è l'indirizzo IP della macchina target.

11.5 Setup Elasticsearch/Kibana

Per il deployment di Elasticsearch e di Kibana si è partiti inizialmente dalla scrittura del file docker-compose.yml qui sotto riportato:

```
version: '3'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.12.0
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - cluster.initial_master_nodes=es01
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
```

```
      - data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    labels:
      kompose.service.expose: "true"
    networks:
      - elastic

  kib01:
    image: docker.elastic.co/kibana/kibana:7.12.0
    container_name: kib01
    ports:
      - 5601:5601
    labels:
      kompose.service.expose: "true"
    environment:
      ELASTICSEARCH_URL: http://es01:9200
      ELASTICSEARCH_HOSTS: '["http://es01:9200"]'
    networks:
      - elastic

volumes:
  data01:
    driver: local

networks:
  elastic:
    driver: bridge
```

Successivamente per orchestrare i container mediante K3s si è utilizzato un tool di conversione da file Docker Compose a risorse Kubernetes denominato Kompose.

Per installarlo su Linux:

```
curl -L
https://github.com/kubernetes/kompose/releases/download/v1.22.0/kompose-linux-
amd64 -o kompose
```

Successivamente andare sulla cartella in cui è presente il file docker-compose.yml e digitare il comando:

```
kompose convert
```

Vengono creati i seguenti 8 file di output: data01-persistentvolumeclaim.yaml, elastic-networkpolicy.yaml, es01-deployment.yaml, es01-ingress.yaml, es01-service.yaml, kib01-deployment.yaml, kib01-ingress.yaml, kib01-service.yaml.

A questo punto per creare la risorsa in un pod a partire dai file di configurazione in formato yaml eseguire il comando:

```
kubectl apply -f data01-persistentvolumeclaim.yaml,elastic-
networkpolicy.yaml,es01-deployment.yaml,es01-ingress.yaml,es01-
service.yaml,kib01-deployment.yaml,kib01-ingress.yaml,kib01-service.yaml
```

Infine è necessario modificare i file es01-deployment.yaml e kib01-deployment.yaml per accedere al pod al di fuori del cluster, in modo tale che il pod possa vedere le interfacce di rete della macchina host su cui esso è in esecuzione (VM in questo caso). Aggiungere quindi il seguente flag nella sezione spec:

```
spec:
  hostNetwork: true
```

11.6 Setup ONOS

Il deployment di ONOS su un cluster K3s parte dalla scrittura del Dockerfile:

```
FROM onosproject/onos
# Ports
# 6653 - OpenFlow
# 6640 - OVSDB
# 8181 - GUI
# 8101 - ONOS CLI
# 9876 - ONOS intra-cluster communication
EXPOSE 6653 6640 8181 8101 9876
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get install -y git
RUN git clone https://gerrit.onosproject.org/onos
RUN cd onos
RUN /bin/bash -c "source onos/tools/dev/bash_profile"
RUN apt-get update && \
    apt-get install maven -y
COPY foo-app /foo-app
RUN cd /foo-app && \
    mvn clean install
ENTRYPOINT ["./bin/onos-service"]
CMD ["server"]
```

La cartella foo-app contiene il codice sorgente del comando CLI e del relativo endpoint REST che implementa la logica per intervenire a fronte dell'attacco di sicurezza definito dallo use case #1, che dovrà poi essere caricato come modulo applicativo in ONOS.

Successivamente va creata un'immagine nel repository locale, denominata onos-image, con il seguente comando da lanciare nella stessa directory in cui è presente il Dockerfile:

```
sudo docker build -t onos-image .
```

L'immagine creata va salvata come file .tar in modo tale da poter essere importata nel cluster K3s.

Per salvare l'immagine come archivio .tar eseguire il comando:

```
sudo docker save onos-image:latest > /<path_to_tar_file>/onos.tar
```

dove <path_to_tar_file> indica la directory in cui sarà salvata l'immagine come file .tar (denominato onos.tar).

A questo punto si può importare l'immagine nel cluster:

```
sudo k3s ctr images import onos.tar
```

Ora si ha tutto l'occorrente per la creazione del pod. Il file .yaml è così definito:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: onos-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      bb: onos
  template:
    metadata:
      labels:
        bb: onos
    spec:
      hostNetwork: true
      containers:
      - name: onos-container
        image: onos-image:latest
        imagePullPolicy: Never
```

Basterà infine lanciare il seguente comando per avviare il pod:

```
kubectl apply -f onos.yaml
```

11.7 Setup CORE

CORE fornisce uno script per aiutare l'utente con l'installazione automatica di tutte le dipendenze:

```
# clone CORE repo
git clone https://github.com/coreemu/core.git
cd core
./install.sh
```

Dopo l'installazione è possibile eseguire core-daemon e core-pygui/core-gui.

CORE permette di eseguire tutti gli switch Ethernet emulati (non nel contesto del singolo nodo) come switch OVS specificando all'esecuzione di core-daemon il flag --ovs:

```
sudo core-daemon --ovs
```

Nella GUI di CORE disporre gli switch nel foglio bianco e far partire l'emulazione. Vengono creati gli switch OVS il cui nome di solito ha un prefisso "b." seguito da una notazione decimale puntata. Per vedere i dettagli degli switch digitare il comando:

```
sudo ovs-vsctl show
```

Gli switch vanno configurati per poter connettersi all'SDN controller. Per far questo configurare gli switch per supportare le versioni di OF, specificare l'indirizzo IP di ONOS (e.g. 192.168.101.5) e la porta in ascolto per le connessioni OF (di default la 6653). Se lo switch stabilisce con successo la connessione con il controller SDN, nella descrizione dello switch appare il flag "is_connected" con valore "true".

Inoltre gli switch devono poter comunicare a L2 con le altre VM e per far questo bisogna aggiungere allo switch le interfacce di rete della VM di CORE. Per semplicità, supponendo che ci sia una VM per ONOS con interfaccia enp0s3, una per Elasticsearch con interfaccia enp0s8 e una per le PMU/PDC con interfaccia enp0s9, lo switch per connettersi punto-punto con queste deve avere le 3 interfacce aggiunte.

Lo script sottostante è un esempio di come deve essere configurato uno switch OVS (con nome b.1.1 in questo caso) nel complesso:

```
#!/bin/bash
sudo ifconfig b.1.1 192.168.101.6/24
sudo ovs-vsctl add-port b.1.1 enp0s3
sudo ovs-vsctl add-port b.1.1 enp0s8
sudo ovs-vsctl add-port b.1.1 enp0s9
sudo ovs-vsctl set bridge b.1.1 protocols=OpenFlow13
sudo ovs-vsctl set-controller b.1.1 tcp:192.168.101.5:6653
```

12 Ringraziamenti

Non si può terminare un percorso senza fare i dovuti elogi a chi mi ha supportato in questi anni universitari.

In primis vorrei ringraziare la mia stupenda famiglia, soprattutto i miei genitori, mia nonna Argentina e mia sorella Silvia, perché senza di loro questa tesi non sarebbe mai stata scritta. Mi hanno sempre appoggiato nei momenti difficili, nelle scelte di vita e creduto in me fino alla fine.

Un grandissimo grazie va al relatore Fulvio Risso, per l'amore che mette nell'attività di insegnamento e la sua capacità di affascinare ed avvicinare gli studenti agli argomenti che tratta a lezione, per i suoi consigli competenti, mi ha corretto ed incoraggiato a migliorarmi là dove necessario.

Grazie al mio tutor Fabrizio Garrone, per avermi dato l'opportunità di mettermi alla prova, per la pazienza e il supporto che mi è stato dato in questi mesi. È stata un'esperienza utilissima per la mia crescita professionale ma soprattutto umana. Grazie ad Andrea Cazzaniga ed a Roberta Terruggia per l'affiancamento in questa fase di apprendimento.

Grazie a Gabriele, Milo e Sergio, per il loro grande aiuto e contributo che ha reso possibile lo svolgimento di questa attività.

Grazie agli amici "storici", gli amici di una vita: Alessia, Davide, Fabio, Federico, Leonardo, i due Luca, Richard, Sebastiano e Sonia. Nonostante la lontananza hanno fatto sentire il loro affetto, rafforzando il grande legame che già c'era.

Grazie agli amici conosciuti in questi anni universitari torinesi: David, Fabrizio, Francesco, Giacomo, Gianluca, Leonardo, Marianna, Nicola, i due Paolo, Rebecca, Riccardo e Vincenzo, per aver condiviso come me ansie e paure, risate e sogni.

Infine grazie a Marialaura, per le brevi passeggiate al lungomare, le lunghe chiacchierate sulla nostra panchina preferita e per tutti i suoi gesti di affetto.

A tutti infiniti ringraziamenti.