ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica — Scienza e Ingegneria

Corso di Laurea Triennale in Informatica

Integrazione di Software-Defined Networking e Infrastructure as Code per la Sicurezza delle Infrastrutture: Analisi e Casi di Studio

Relatore:
Saverio Giallorenzo

Presentata da:
Filippo Marini

Correlatore:
Simone Melloni

Sessione III

Anno Accademico 2024/2025

Indice

Capitolo 1

Introduzione

Negli ultimi anni, le infrastrutture IT hanno iniziato ad adottare sempre più modelli che favoriscono l'automazione, la scalabilità e la gestione centralizzata. Le reti tradizionali, basate su configurazioni manuali e sistemi hardware dedicati, presentano molte limitazioni in termini di flessibilità, velocità di attuazione di modifiche e l'introduzione di nuovi sistemi, e il controllo della sicurezza. Analogamente, la gestione manuale delle infrastrutture, come reti aziendali e datacenter fino alle infrastrutture cloud, presentano evidenti limiti nella velocità di provisioning e manutenzione delle risorse, nella scalabilità e soprattutto nella gestione di errori di configurazione, che implicano a volte anche potenziali rischi di sicurezza.

Per questo motivo, i paradigmi del Software-Defined Networking e dell'Infrastructure as Code, illustrati nel Capitolo 2, vengono adottati per introdurre miglioramenti significativi dal punto di vista di gestione e di sicurezza, grazie all'utilizzo di tecnologie che permettono la realizzazione automatica di configurazioni, rispettivamente di rete e infrastrutture, tramite software.

Il Software-Defined Networking permette la divisione del piano di controllo dal piano di dati, o di inoltro, normalmente uniti all'interno di un singolo dispositivo di rete, permettendo quindi una gestione automatizzata di una rete, utilizzando dei software e applicazioni di alto livello per le configurazioni dei singoli sistemi, migliorando così anche il controllo generale della sicurezza.

L'Infrastructure as Code, invece, introduce un approccio di gestione di un'infrastruttura basato su codice che definisce lo stato voluto dell'infrastruttura stessa, piuttosto che l'esecuzione manuale dell'approvigionamento di ogni singola risorsa. Questo tipo di architettura sfrutta l'utilizzo di software specifici, per la definizione del contenuto di un'infrastruttura, per eventuali modifiche da applicare ad essa e per poterla replicare in ambienti diversi.

La gestione automatica tramite definizione di codice permette di creare delle versioni

di configurazione (di rete o infrastruttura), in cui ogni versione rappresenta uno stato diverso, ma soprattutto riduce notevolmente l'intervento manuale, eliminando così eventuali errori che possono accadere nella realizzazione manuale delle configurazioni e che possono esporre potenziali vulnerabilità di sicurezza.

L'integrazione di questi due modelli permette la realizzazione di infrastrutture completamente "software-defined", in cui un'intera architettura distribuita, costituita dalle risorse infrastrutturali e la topologia di rete che le mette in comunicazione, può essere definita, versionata e gestita tramite codice.

Il lavoro della presente tesi riguarda l'integrazione di tali paradigmi per la creazione di architetture distribuite definite da software nella piattaforma Proxmox attraverso l'utilizzo di tecnologie SDN e IaC. In particolare, vengono esplorati casi di studio nel Capitolo 4 mostrando tre tipi incrementali di segmentazione di una rete di tecnologia operazionale, creando tre definizioni diverse della stessa rete tramite codice dichiarativo. Lo studio evidenzia come le caratteristiche di automazione e programmabilità rendano la definizione di un'infrastruttura riutilizzabile, modificabile e applicabile automaticamente tramite software appositi. Il progetto open-source è reperibile nel repository Git "https://github.com/Flooding-against-Ransomware/OT_case_studies"

Nel Capitolo 3, invece, vengono anche illustrati i vantaggi e gli svantaggi generali di sicurezza introdotti dalle due architetture, mostrando i miglioramenti sostanziali di entrambi, anche con l'ausilio di esempi, insieme ai principali rischi di sicurezza da cui sono affetti e le vulnerabilità specifiche che essi introducono.

Infine, nel Capitolo 5 viene effettuato un riepilogo dei contributi di ricerca e di sviluppo, rispettivamente al Capitolo 3 e al progetto del Capitolo 4, della presente tesi e discussi eventuali direzioni future per l'espansione dei lavori presentati.

Capitolo 2

Preliminari

Il Software Defined Networking è un paradigma che punta a virtualizzare le risorse di una rete fisica classica. Lo scopo di una Software-Defined Network (SDN) è quello di creare una rete che sia gestibile e manutenibile centralmente e che sia programmabile, quindi configurabile attraverso software, anche in modo automatico.

Il vantaggio di avere un'infrastruttura di rete virtuale sta nella gestione delle risorse. In una rete fisica, ogni risorsa è costituita da una macchina hardware che viene gestita e configurata manualmente e singolarmente. Inoltre, si tratta anche di una questione di spazio e di costo: dal momento che le risorse sono fisiche, esse occupano spazio all'interno dell'infrastruttura e bisogna anche affrontare il costo di acquisto di nuove risorse hardware per eventuali espansioni dell'infrastruttura stessa. Al contrario, in un'infrastruttura virtuale gestita tramite software, le risorse di rete possono essere rimosse o aggiunte virtualmente in modo automatico, di conseguenza molto più velocemente, permettendo contemporaneamente la gestione di più risorse occupando anche significativamente meno spazio fisico, dal momento che ogni nodo può ospitare molteplici risorse virtuali, riducendo così anche il costo hardware. Questa gestione puramente automatica e centralizzata permette la manipolazione delle risorse virtuali senza alcuna interazione fisica da parte degli amministratori di rete.

Questo concetto di automazione rende perfette le reti virtuali per paradigmi presenti nel Cloud come "Infrastructure as Code (IaC)" [1] che è essenzialmente un modello concettuale che permette la gestione e il provisioning di un'infrastruttura tramite l'utilizzo di codice, piuttosto che manualmente, la spiegazione dettagliata del paradigma IaC è presente nella sezione 2.3. Per "provisioning" [2] si intende il processo di creazione, configurazione e dispiegamento di risorse, come server, router e switch, che insieme compongono un'infrastruttura "Information Technology (IT)". Nell'IaC, il provisioning permette di gestire in modo automatico e scalabile la configurazione di un'infrastruttura virtuale che si appoggia fisicamente su un cluster di computer.

"Proxmox Virtual Environment" [3], o Proxmox VE, è una piattaforma open-source di gestione server per la "virtualizzazione d'impresa" che permette la virtualizzazione di sistemi operativi attraverso macchine virtuali e container, utilizzando rispettivamente "KVM Hypervisor" e "Linux Containers (LXC)", e soprattutto la virtualizzazione della rete, mettendo a disposizione l'integrazione di una SDN interna comune a tutto il cluster. Proxmox VE è accessibile e gestibile tramite l'utilizzo di un'interfaccia web di controllo, dalla quale si possono monitorare e configurare tutti i nodi fisici del cluster, decidendo, ad esempio, su quali di essi dispiegare le macchine virtuali oppure definire quali nodi fisici apparterranno a determinate zone della rete virtuale. Il funzionamento dettagliato di Proxmox VE viene mostrato nella sezione 2.2.

Proxmox fornisce anche delle "REST API" tramite le quali è possibile interagire con l'intero cluster e che permettono di impostare la maggior parte delle configurazioni che possono essere eseguite manualmente dall'interfaccia web di controllo. Queste API possono essere utilizzate per eseguire il provisioning di una determinata infrastruttura definendo il codice opportuno.

Il software utilizzato, presentato nella sezione 2.4, è Terraform che, tramite determinati provider, sfrutta l'utilizzo delle API Proxmox per automatizzare il provisioning di risorse come macchine virtuali e le risorse che compongono l'intera SDN, il tutto direttamente ed esclusivamente tramite codice dichiarativo.

2.1 Struttura e Funzionamento di una SDN

Questa sezione espone in modo tecnico le componenti e il funzionamento del paradigma SDN.

Come anticipato precedentemente, il Software Defined Networking [4] è un modello di architettura di rete che punta a virtualizzare le risorse di rete, controllandole centralmente tramite l'utilizzo di software. In particolare, una SDN separa le funzioni di controllo dalle funzioni di inoltro del traffico di rete presenti in ogni nodo dell'infrastruttura. Essi sono i cosiddetti "Control Plane" e "Data Plane" e, a differenza di una rete tradizionale che mantiene entrambi i piani all'interno di ogni singolo dispositivo di rete, come gli switch o i router, in una SDN questi due piani vengono separati. In particolare, il piano dati risiede ancora all'interno dei nodi della rete, mentre il piano di controllo, che è proprio la parte responsabile della logica decisionale per l'instradamento dei pacchetti di rete, viene spostato all'interno del "Controller", un componente della SDN centralizzato e gestito unicamente tramite software. In questo modo, il modello di Software Defined Networking si allinea con i concetti di Infrastructure as Code, poichè l'intera rete può

essere configurata, programmata e orchestrata automaticamente attraverso l'utilizzo di software.

Riguardo al funzionamento di una rete, il termine "Plane" [5], cioè piano o superficie, descrive uno specifico strato dell'architettura di rete, come appunto il piano di controllo e il piano dati. Queste superfici non sono dei veri e propri componenti fisici, ma piuttosto esprimono un concetto che serve come distinzione e che aiuta a capire come effettivamente fluisce il traffico dati all'interno di una rete.

2.1.1 Control Plane

Il Control Plane è lo strato di rete che si occupa di determinare le rotte per i vari tipi di traffico, di compilare le tabelle di forwarding dei singoli dispositivi di rete e definire le policy che devono essere seguite, il tutto per definire e gestire l'instradamento del traffico dati all'interno della rete. Si può affermare che il piano di controllo sia il responsabile delle decisioni ad alto livello riguardo alle configurazioni generali della rete. Come già anticipato, nel paradigma SDN, il control plane è implementato tramite software e non risiede più all'interno dei singoli nodi di rete, ma è totalmente separato e centralizzato all'interno di un unico componente chiamato "Controller". Questo componente, descritto dettagliatamente in seguito, è capace di comunicare con l'intera rete sottostante e si occupa della programmazione e dell'invio delle regole di routing ai nodi. Il punto di avere il control plane all'interno di un singolo componente di rete permette il controllo centralizzato, piuttosto che provvedere alla configurazione manuale direttamente all'interno di ogni singolo dispositivo.

2.1.2 Data Plane

Il Data Plane, anche conosciuto come "forwarding plane", è il responsabile dell'instradamento effettivo dei pacchetti di rete in base alle regole definite e ricevute dal Control Plane. Nel modello SDN, il Data Plane è implementato all'interno di dispositivi di rete, come ad esempio gli switch o i router, ai quali ci si riferisce solitamente con il nome di "forwarding elements". Questi dispositivi fanno affidamento esclusivamente sulle istruzioni ricevute dal Control Plane, in altre parole dal Controller, cioè si occupano di processare ogni pacchetto in ingresso in base alle regole di instradamento che sono state definite dal Controller e inviate a tutti i dispositivi della rete. Il piano dati è tanto importante quanto il piano di controllo, poichè se quest'ultimo è responsabile dell'intera gestione e configurazione della rete, come descritto prima, il piano dati invece è proprio quello che esegue l'instradamento effettivo dei dati all'interno della rete stessa.

Guardando questo concetto da un punto di vista dei dispositivi, il Controller, che rappresenta il Control Plane, è responsabile della definizione delle "istruzioni" relative al traffico dati, mentre i dispositivi di rete, quali switch e router, che rappresentano invece il Data Plane, sono quelli che ricevono tali istruzioni e agiscono di conseguenza, cioè eseguono il routing del traffico di rete e rispettano le policy basandosi sulle regole fornite dal Controller.

2.1.3 Interfacce Northbound e Southbound

Come descritto nella precedente sottosezione, avviene una comunicazione cruciale tra il Control e il Data plane, più nello specifico, tra i dispositivi che li rappresentano. In questa sottosezione viene illustrato come avviene tale comunicazione.

La SDN sfrutta l'utilizzo di due tipi di API per permettere la comunicazione attraverso gli strati di controllo e dei dati e la comunicazione più ad alto livello per poter gestire il Controller della SDN. Questi due tipi di interfacce sono il "Southbound" e il "Northbound" API.

Southbound API - Le Southbound API sono utilizzate per permettere la comunicazione tra il Control Plane e il Data Plane. Tramite l'utilizzo di questo tipo di interfaccia, il Controller è in grado di comunicare direttamente con i nodi della rete sottostante per poter definire e configurare le loro impostazioni di rete, può recuperare le informazioni relative alla topologia e allo stato dell'intera rete e ricevere notifiche riguardo ad eventi come eventuali problemi di collegamento tra due dispositivi oppure a delle congestioni tra determinati nodi. Southbound API comuni includono, ad esempio, il protocollo "OpenFlow", che è largamente utilizzato per la comunicazione tra il Controller e gli switch di rete.

Northbound API - Le Northbound API, invece, sono utilizzate per esporre le funzionalità del Controller della rete ad applicazioni e servizi di gestione di rete di livello superiore. Questo tipo di interfaccia permette ad applicazioni esterne di interagire direttamente con il Controller della SDN, cioè permette di richiedere dei servizi di rete e ottenere informazioni relative alla topologia, al flusso di dati tra i nodi interni ed eventualmente anche delle metriche per le perfomance della rete. Le Northbound API offrono quindi la "programmabilità" e l'automazione della gestione di rete, abilitando anche l'integrazione con software di orchestrazione, piattaforme cloud e altri strumenti di gestione.

2.1.4 Controller

Come descritto in precedenza in questa sezione, il Controller è il componente centrale nel modello di Software Defined Networking, responsabile dell'implementazione delle funzioni di controllo della rete e del coordinamento delle comunicazioni tra il Control Plane e il Data Plane, attraverso l'interfaccia Southbound [6]. Questo componente è il motivo principale per cui il modello SDN può essere programmabile e gestibile in modo automatico. Infatti, proprio grazie alle API Northbound, è possibile comunicare con esso attraverso applicazioni o sistemi di gestione di alto livello chiamati sistemi di "Management and Orchestration (MANO)", responsabili del provisioning e della configurazione e monitoraggio delle risorse e del traffico di rete. Questi componenti sono utilizzati anche per eseguire specifici compiti, per l'ottimizzazione dell'utilizzo di risorse, per eseguire l'eventuale "load balancing" del traffico interno e di assicurare la continuità e disponibilità dei servizi di rete. Dopodichè, tramite le API Southbound, tutte le varie istruzioni definite dai sistemi "MANO" vengono inviate dal Controller ai singoli nodi sottostanti, utilizzando specifici protocolli come "OpenFlow", in modo tale da attuare e rendere effettivo il dispiegamento dell'intera rete. In questo modo, di fatto, è possibile disporre e modificare l'intera infrastruttura di rete in modo totalmente automatico e centralizzato, eliminando quindi le singole configurazioni manuali e ripetitive che si dovrebbero altrimenti eseguire su ogni nodo dell'intera rete. Grazie alla comunicazione bidirezionale con la rete sottostante, il Controller permette anche una gestione relativa alla sicurezza della rete. Infatti, ad esempio, se all'interno di un'infrastruttura sono presenti dei sistemi di sicurezza che eseguono il monitoraggio continuo del traffico di rete, è possibile configurare la comunicazione da questi sistemi verso il Controller stesso, in modo tale che quest'ultimo possa attuare misure di sicurezza in base agli eventi che riceve da tali sistemi. Il lato di sicurezza di rete che il modello di SDN è in grado di offrire viene illustrato nel Capitolo 3, mostrando anche i vantaggi e gli svantaggi rispetto a una rete fisica tradizionale.

2.2 Proxmox Virtual Environment

Come anticipato, Proxmox VE è una piattaforma di virtualizzazione basata su "Debian Linux" e completamente open-source, che integra risorse di computazione, di storage e di rete. Inoltre, permette di gestire cluster "Highly Available", cioè cluster con il minimo disservizio, così come backup e disaster recovery. Tutti i componenti al suo interno sono "Software-Defined" e compatibili l'uno con l'altro. [7].

Proxmox VE, è stato progettato per il provisioning di "Hyper-Converged Infrastructures (HCI)" [8]. Le infrastrutture iperconvergenti [9] sono essenzialmente delle infrastrutture Software-Defined che puntano a virtualizzare tutti gli elementi di un sistema convenzionale basato su hardware. Queste infrastrutture comprendono la virtualizzazione di sistemi operativi, quindi le macchine virtuali tramite l'utilizzo di "hypervisor" di tipo 1 nel caso di Proxmox, "Software-Defined Storage", ovvero la virtualizzazione dello storage utilizzata per le macchine virtuali, e infine la virtualizzazione di rete tramite Software-Defined Networking.

L'interfaccia web integrata di Proxmox [7], accessibile da ogni nodo, permette una gestione centralizzata dell'intero cluster, dei nodi al suo interno, delle macchine virtuali, dei container e anche della rete virtuale del datacenter. È possibile anche visualizzare i log di sistema di tutti i nodi presenti nel cluster, in modo da monitorare ogni evento ed errore che si verifica a seguito di determinate azioni. Le macchine virtuali e i container possono essere avviate direttamente dall'interfaccia web attraverso l'utilizzo di una console interattiva che rappresenta l'interfaccia grafica della macchina stessa. Sono anche presenti dei sistemi di monitoraggio delle risorse sia per i singoli nodi sia per le macchine virtuali, tramite i quali si può monitorare l'utilizzo di risorse da parte delle macchine, come ad esempio l'utilizzo del processore, della RAM e dello spazio di archiviazione disponibile e occupato.

Proxmox fornisce il "Cluster File System (pmxcfs)", un file system condiviso basato su database utilizzato per memorizzare i file di configurazione. In questo modo, è possibile memorizzare la configurazione di ogni macchina virtuale o container creati sui nodi. Utilizzando il software "Corosync", creato per la comunicazione di gruppo, tutte le configurazioni vengono inviate in tempo reale a tutti i nodi del cluster, mentre il file system condiviso le memorizza all'interno di un database su disco.

All'interno del cluster è possibile definire dei permessi basati su ruoli da assegnare agli utenti. In questo modo, è possibile assegnare determinati privilegi e gestire e controllare gli accessi ad oggetti del cluster. Ogni permesso definisce un soggetto, un utente o un gruppo, e un ruolo, cioè un insieme di privilegi. Infine, Proxmox gestisce le autenticazioni degli utenti tramite i suoi "Authentication Realms". Attualmente, Proxmox supporta molteplici fonti di autenticazione come "Active Directory" di Microsoft, LDAP

(Lightweight Directory Access Protocol), Linux Pam, essendo Proxmox una distribuzione Linux, oppure il sistema di autenticazione integrato di Proxmox stesso.

Infine, Proxmox mette a disposizione delle REST API tramite le quali è possibile eseguire tutte le operazioni sul cluster che possono essere effettuate dall'interfaccia web. Queste API sono contattabili attraverso richieste HTTP e possono essere sfruttate da diverse applicazioni, nel caso di questo studio verrà utilizzato il software "Terraform".

2.2.1 Proxmox SDN

La piattaforma Proxmox Virtual Environment offre anche una funzionalità di virtualizzazione di rete attraverso la sua Software-Defined Network, che permette la creazione di zone virtuali e semplifica e automatizza configurazioni avanzate di rete. [10]

La SDN di Proxmox gesisce la separazione di rete attraverso l'utilizzo di zone, un'area propria e virtualmente separata a cui possono essere assegnati dei nodi del cluster, la VNet, una rete virtuale che appartiene ad una zona, ed infine una subnet, che definisce un range di indirizzi ip all'interno di una VNet. Il tipo della zona che viene creata nella SDN può determinare un diverso comportamento della rete, offrendo anche specifiche funzionalità, vantaggi e limitazioni.

La SDN può essere configurata per ottenere delle semplici reti virtuali separate e appartenenti a ciascun nodo fino a poter configurare delle intere reti complesse di overlay, cioè delle reti virtuali che si appoggiano su una rete fisica reale e che comprendono molteplici nodi oppure l'intero cluster.

Come già anticipato, la configurazione della SDN può essere eseguita tramite l'utilizzo dell'interfaccia web integrata di Proxmox, la quale offre una vista completa dell'intera rete virtuale, garantendo la possibilità di creare nuove zone, VNet, subnet, range di indirizzi IP, offrendo anche l'utilizzo del DHCP per l'assegnamento automatico degli indirizzi agli host virtuali. L'interfaccia permette anche di controllare e visionare tali impostazioni tramite l'IPAM (IP Address Management), che offre una vista su tutto ciò che è stato creato all'interno della SDN, come ad esempio i range di indirizzi IP utilizzati, la disposizione delle VNet e delle subnet al loro interno e anche le macchine virtuali assegnate ad esse.

Ogni configurazione della SDN viene memorizzata in "/etc/pve/sdn" e condivisa con tutti i nodi del cluster grazie al Proxmox Cluster File System. Successivamente quelle configurazioni vengono tradotte nei rispettivi formati di configurazione dei tool presenti in ogni nodo, ad esempio, le regole del firewall dei nodi vengono tradotte in regole IPTables.

Zona

Una zona definisce una rete virtualmente separata e ristretta ai nodi che gli vengono assegnati, definiti da un attributo al momento della creazione della zona stessa. Inoltre, è possibile definire dei permessi in modo da limitare gli utenti ad una determinata zona e alle VNet e subnet che essa contiene.

Proxmox fornisce cinque tipi di zone differenti:

Simple Zone - È il plugin più semplice tra tutti e crea un bridge VNet isolato. Non è collegata a nessuna interfaccia fisica e il traffico di rete della macchine virtuali può essere solamente interno al nodo a cui è assegnata. Tuttavia, è comunque possibile abilitare il NAT sorgente (Network Address Translation) all'interno delle subnet associate alla zona per poter instradare il traffico interno verso reti esterne, potenzialmente verso gli altri nodi del cluster.

VLAN Zone - Questo plugin utilizza un bridge Linux locale per connettersi all'interfaccia fisica locale del nodo. Questo tipo di zona utilizza il tagging definito all'interno della VNet assegnata per poter isolare i segmenti di rete. In questo modo permette alle macchine virtuali dispiegate su nodi differenti di comunicare tra loro, se appartenenti allo stesso tag VLAN. Per utilizzare questo tipo di zona è richiesto un supporto fisico esterno adatto, cioè uno switch che supporti la segmentazione VLAN.

QinQ Zone - Anche conosciuta come "VLAN Stacking", utilizza molteplici livelli di tag VLAN per l'isolamento. Questa zona definisce il tag VLAN esterno mentre quello interno è definito dalla VNet assegnata. Per poter utilizzare questo tipo di zona è richiesto che il supporto fisico esterno, cioè lo switch, supporti le "stacked VLANs", cioè VLAN a più livelli.

VXLAN Zone - Questo tipo di zona stabilisce un tunnel come "overlay" al di sopra di una rete fisica "underlay". In questo modo è possibile incapsulare frame Ethernet di livello 2 in pacchetti UDP di livello 4 per poter comunicare con la rete esterna. Per far sì che i peers, cioè le macchine, che si desideri mettere in comunicazione possono farlo attraverso questo tipo di connessione, bisogna abilitare ed accertarsi che sia attiva la connettività UDP su ognuno di essi, altrimenti i pacchetti UDP contenenti i frame VXLAN verranno semplicemente scartati dal peer destinatario. Questo tipo di zona permette, grazie alla metodologia appena illustrata, di poter creare una VXLAN overlay al di sopra dell'intera Internet, permettendo quindi di poter far fuoriuscire il traffico dal cluster, comunicando con nodi esterni, non necessariamente altre macchine virtuali o nodi Proxmox, come se tutti i nodi o

macchine virtuali condividano la stessa rete di livello 2, cioè la Local Area Network (LAN). Non è richiesto l'utilizzo di uno switch per questo tipo di zona siccome si affida a un tipo di traffico instradabile a livello 4.

EVPN Zone - Quest'ultimo tipo di zona è in grado di creare una rete di livello 3 instradabile, capace anch'essa di distribuirsi attraverso molteplici cluster. Questo è possibile grazie all'utilizzo di una connessione VPN che utilizza il "Border Gateway Protocol (BPG)" come protocollo di instradamento del traffico. La VNet assegnata può possedere un indirizzo IP o MAC di tipo "anycast", cioè uno stesso indirizzo che può essere assegnato a più nodi sulla rete. Inoltre, questa zona supporta anche l'utilizzo di un Controller fornito da Proxmox tramite plugin.

VNet

Una rete virtuale (VNet) è essenzialmente un bridge linux virtuale che viene assegnato ad una zona è funge da contenitore per una o più subnet. Una volta creata una VNet tramite l'interfaccia web, viene definita una nuova interfaccia virtuale locale su ogni nodo del cluster, secondo Proxmox [10]. Successivamente, si può creare una macchina virtuale alla quale è possibile assegnare la VNet come scheda di rete virtuale, che verrà mostrata come bridge disponibile del nodo. A quel punto, tale macchina virtuale sarà connessa a quella rete virtuale. Inoltre, se viene definita una subnet associata alla VNet, la macchina virtuale, a cui è stata assegnata la VNet come scheda di rete, apparterrà anche a tale subnet e pertanto è possibile assegnarle un indirizzo ip, o manualmente oppure tramite DHCP se abilitato dalla zona in cui la VNet e la subnet risiedono. Durante la creazione della zona sull'interfaccia web, è possibile impostare anche alcuni attributi che permettono l'abilitazione della VLAN in quella rete virtuale. Infatti, se si imposta l'attributo "VLAN Aware" e si definisce un numero per l'attributo "Tag", quella VNet sarà in grado di riconoscere e utilizzare traffico VLAN con il tag definito. In questo modo, le macchine virtuali possono effettivamente comunicare tramite connessione VLAN.

Subnet

Una subnet definisce uno specifico range di indirizzi IP descritto tramite l'utilizzo del modello CIDR. Una VNet può contenere una o più subnet all'interno delle quali deve essere specificato l'indirizzo IP della subnet stessa, insieme a quello del default gateway, inoltre può essere anche specificato un range di indirizzi IP, che può essere utilizzabile dal DHCP, se abilitato, oppure per restringere gli indirizzi che possono essere utilizzati all'interno della VNet. Come è stato anticipato nella spiegazione della zona Simple, in una subnet può essere abilitato il Source Network Address Translation, tramite l'attributo

"snat" durante la creazione della subnet, che permette di instradare il traffico al di fuori del nodo su cui si trova la zona di livello 3. Infine, è anche possibile definire un prefisso DNS da assegnare nel dominio di registrazione.

2.3 Infrastructure as Code: Tecnologie, Linguaggi e Casi d'Uso

"Infrastructure as Code" è un paradigma che esegue il provisioning automatico di un'intera infrastruttura virtuale tramite la definizione di codice, piuttosto che manualmente. Praticamente, vengono definiti dei file di configurazione che descrivono come l'infrastruttura deve essere, cioè quali risorse deve contenere e come queste si relazionano tra di loro. In questo modo, un file di configurazione di un'infrastruttura definisce lo stato, attuale o applicabile in base alle necessità, dell'infrastruttura stessa [1].

2.3.1 Funzionamento

IaC unisce due elementi chiave che sono il workflow, ovvero il ciclo di lavoro, e gli strumenti automatizzati [11]. In questo modo, permette di creare una automatizzazione del provisioning, definendo i file di configurazione e generando gli ambienti virtuali su cloud o on-premise.

Il workflow definisce tutto il processo, che parte dalla definizione dei requisiti dell'infrastruttura alla distribuzione vera e propria delle risorse e si divide in quattro fasi:

Scrittura - Gli sviluppatori creano i file di configurazione che contengono la definizione dell'infrastruttura, cioè le risorse che devono essere create e le modalità di configurazione. Questi file vengono creati solitamente in ambienti di sviluppo integrati che contengono dei controlli di errori sintattici. Il processo di scrittura dei file di configurazione non deve contenere errori, altrimenti un errore nella definizione dell'infrastruttura risulterebbe ad esempio nella creazione di una risorsa in più oppure nell'errata definizione di una di esse.

Controllo della Versione - I file di configurazione, creati nel processo precedente, vengono memorizzati in sistemi di controllo delle versioni come, per esempio, Git. In questo modo, è possibile tenere traccia delle varie modifiche a un'infrastruttura permettendo di eseguire il "rollback" ad una versione precedente in caso di errori o altre problematiche all'interno dell'infrastruttura.

Provisioning - Questo è il processo in cui vengono letti i file di configurazione, da un motore interno, e successivamente create le risorse, come ad esempio delle macchine

virtuali, delle risorse di rete, delle definizioni di ruoli o regole per i firewall, definite all'interno del file stesso, rispecchiando come l'infrastruttura deve essere realmente. Questo processo è "idempotente", vuol dire che, nel caso dell'IaC, leggendo la stessa versione di un file di configurazione, verrà creata sempre la stessa infrastruttura con le stesse risorse e configurazioni, indipendentemente dal linguaggio in cui questi file vengono scritti.

Distribuzione - Infine, nell'ultimo processo l'infrastruttura viene distribuita attraverso i vari ambienti, come l'ambiente di sviluppo, di test e di produzione, mantenendo la coerenza tra essi, cioè deve essere identica in tutti gli ambienti, grazie all'utilizzo del provisioning automatico. Se l'infrastruttura è identica in tutti gli ambienti allora è possibile applicare il cosiddetto "continuos deployment", ovvero una strategia per la produzione software che permette di fare test automatici sulle modifiche del codice e che, una volta passati i test, vengono applicate direttamente al software in produzione. Il concetto nel paradigma dell'IaC è lo stesso, a patto che l'infrastruttura sia identica in tutti gli ambienti poichè in questo modo se l'infrastruttura funziona o fallisce nell'ambiente di test, funzionerà o fallirà nello stesso modo nell'ambiente di produzione o di sviluppo.

Gli strumenti automatizzati, invece, sono quelli che forniscono i meccanismi tecnici per poter definire e tracciare le versioni e creare l'infrastruttura tramite codice.

File di Configurazione - Come già illustrato, i file di configurazione sono quelli che definisco l'effettiva infrastruttura. Questi file rappresentano il progetto generale su cui si deve basare l'infrastruttura. In particolare, indicano quante risorse devono essere create, le loro configurazioni, le impostazioni della rete, la definizione di ruoli ecc., e sono scritti in un formato che può essere letto e poi eseguito dagli strumenti di automazione. Questi file, come anticipato prima, sono in grado di mantenere la coerenza tra le istanze della stessa infrastruttura nei vari ambienti e inoltre possono essere modificati in base alle necessità, ovvero se l'aggiornamento di un'infrastruttura richiede l'aumento o la diminuzione di risorse, la definizione di nuove regole o ruoli e così via.

Sistemi di Controllo delle Versioni - Questi sistemi sono in grado di memorizzare la cronologia di ogni file di configurazione, ovvero tutte le modifiche effettuate nel corso dello sviluppo o aggiornamento dell'infrastruttura, da chi e quando sono state eseguite. In questo modo è possibile recuperare eventuali modifiche precedenti che sono state eliminate per poterle ripristinare, oppure permette di identificare, nel caso ci siano eventuali problemi, quale modifica ha portato ad un errore per poter

ripristinare lo stato precedente funzionante, in altre parole eseguire il rollback della versione dell'infrastruttura.

Motori di Automazione - Essi hanno il compito di automatizzare il provisioning basandosi sui file di configurazione che ricevono. Questi sistemi sono proprio i responsabili della creazione effettiva e finale delle risorse e, più in generale, dell'istanziazione dell'intera infrastruttura. In pratica, leggono le definizioni dell'infrastruttura scritte nel file di configurazione trasformando il codice in risorse effettive come macchine virtuali, container, risorse di rete.

Integrazione di API e Piattaforme - Molto spesso, gli strumenti IaC si interfacciano con piattaforme cloud attraverso le "Application Programming Interface (API)". Tramite queste API, gli strumenti di IaC possono comunicare direttamente con i servizi cloud per garantire appunto il provisioning automatico. Questo è proprio il concetto sui cui si basa il caso di studio di questa tesi, ovvero utilizzare uno strumento automatizzato, cioè Terraform, che sfrutta le API della piattaforma Proxmox, per poter eseguire provisioning automatico tramite codice della sua Software-Defined Network, insieme anche a macchine virtuali e regole per i firewall.

2.3.2 Approcci dell'IaC

Prima della creazione di un'infrastruttura tramite codice, vengono definite alcune scelte. In particolare, viene chiarita la metodologia di definizione dell'infrastruttura, se in modo dichiarativo o imperativo, e anche se essa può essere modificata oppure no a seguito della sua creazione. Per la definizione del codice di un'infrastruttura si utilizzano due metodologie principali che sono quella dichiarativa e quella imperativa.

Approccio Dichiarativo - È il metodo più comune per definire un'infrastruttura, anche chiamato "IaC dichiarativo". Viene descritto lo stato desiderato dell'infrastruttura che si vuole ottenere, cioè la definizione effettiva del file di configurazione come descritto prima, successivamente sarà lo strumento automatizzato ad occuparsi dell'implementazione ed eventualmente delle comunicazioni con le API del servizio. Solitamente, lo strumento utilizzato memorizza localmente un file di stato dell'infrastruttura per tenere traccia appunto dello stato in cui si trova attualmente l'infrastruttura, rendendo anche l'arresto o il "deprovisioning", cioè la rimozione dell'infrastruttura, più veloce [1]. Nella sezione successiva verrà illustrato come Terraform mantiene localmente un file di questo tipo per tracciare lo stato dell'infrastruttura che ha creato, in particolare il numero e tipo di risorse e le configurazioni di ognuna.

Approccio Imperativo - Questo approccio, chiamato anche approccio procedurale, richiede la definizione degli specifici comandi necessari al provisioning dell'infrastruttura e anche l'ordine in cui essi devono essere eseguiti [1]. Questo approccio è molto più complesso e facilita anche gli errori nella definizione del file di configurazione, pertanto richiede una certa esperienza.

Generalmente, quando si apportano delle modifiche alla configurazione dell'infrastruttura, nell'approccio dichiarativo queste modifiche vengono applicate automaticamente dagli strumenti automatizzati che rilevano dei cambiamenti rispetto al loro file di stato, mentre l'approccio imperativo richiede che le modifiche siano applicate manualmente [1].

Un'infrastruttura può inoltre essere mutabile o immutabile, cioè a seguito del provisioning le modifiche possono essere direttamente applicate all'infrastruttura dispiegata oppure no.

Infrastruttura Mutabile - Questo tipo di infrastruttura permette le modifiche ad hoc per la risoluzione o l'aggiunta di eventuali requisiti che l'infrastruttura deve possedere oppure per aggiungere delle modifiche di sicurezza immediate. In pratica, le modifiche vengono applicate dopo che il provisioning è stato eseguito e senza dover effettuare il deprovisioning. Questo vuol dire che le modifiche vengono applicate direttamente all'infrastruttura senza doverla rimuovere dal suo ambiente. Tuttavia questo può portare a problemi di coerenza tra gli ambienti, poichè se si mantiene la stessa istanza di un'infrastruttura in diversi ambienti, allora le modifiche andranno applicate ad ognuno di essi per poter mantenere la coerenza tra le varie istanze.

Infrastruttura Immutabile - il modello immutabile, invece, non permette alcuna modifica all'infrastruttura quando essa è stata dispiegata. Ciò significa che se è necessaria una qualche modifica ad alcune configurazioni definite nei file, allora bisogna rimuovere l'infrastruttura dall'ambiente, cioè eseguire il deprovisioning, definire le modifiche nel file di configurazione e poi dispiegare nuovamente l'infrastruttura, quindi rieseguire il provisioning. Questo metodo può non sembrare la soluzione migliore, tuttavia risolve alcuni problemi della mutabilità, come ad esempio il fatto di mantenere la coerenza tra gli ambienti, poichè le modifiche sono scritte direttamente nel file di configurazione e successivamente applicato negli ambienti ad esempio di sviluppo, test e produzione, pertanto non è richiesta la gestione di modifiche singole e manuali e permette di mantenere delle versioni dell'infrastruttura, siccome ogni modifica determina una specifica istanza con versione, facilitando così anche eventuali operazioni di rollback. Tuttavia, rimuovere e riapplicare l'infrastruttura ad ogni versione può comportare anche un certo costo

in termini di tempo, poichè per infrastrutture con numerose risorse il dispiegamento non è istantaneo e dipende anche se il provisioning viene eseguito on-premise oppure su cloud.

2.3.3 Vantaggi e Difficoltà all'Adozione dell'IaC

Tradizionalmente, il dispiegamento, la configurazione e la gestione di un'infrastruttura vengono eseguiti manualmente su ogni singola risorsa fisica presente. Tuttavia, al giorno d'oggi la gestione generale di un'infrastruttura viene effettuata sfruttando tecnologie come la virtualizzazione, che ad esempio permette di avere più risorse virtuali su un'unica macchina fisica insieme a tanti altri vantaggi, e modelli come il cloud computing che permette ad aziende e compagnie di delegare la gestione e la sicurezza della propria infrastruttura a un servizio cloud [1].

L'automatizzazione della gestione dell'Infrastructure as Code permette molteplici benefici [11]. In generale il provisioning manuale di un'infrastruttura richiede molto tempo e competenze specifiche, di conseguenza anche un personale specializzato. L'IaC, invece, essendo gestito tramite software, permette di eseguire il provisioning automaticamente e in brevissimo tempo, consentendo agli sviluppatori di eseguire script per implementare le risorse in poco tempo. Inoltre, viene aumentata anche l'affidabilità [12], siccome ad ogni provisioning dell'infrastruttura il risultato sarà identico in qualsiasi ambiente venga dispiegata. Questo perchè, come già ampiamente mostrato, l'automatizzazione avviene leggendo dei file di configurazione che contengono la descrizione di come l'infrastruttura deve essere. Pertanto, se si utilizza sempre la stessa versione di configurazione, ogni volta verrà creata la stessa infrastruttura. Questo permette anche di evitare errori manuali di configurazione, come ad esempio errori sintattici di scrittura o esecuzione di comandi oppure impostazioni sbagliate, che sono quasi inevitabili, sia critici che non, se viene effettuato il provisioning manualmente su ogni singola risorsa dell'infrastruttura.

Altri vantaggi sono ad esempio la scalabilità che permette di effettuare modifiche direttamente sull'infrastruttura e in tempi molto brevi, permettendo delle eventuali ma anche momentanee espansioni durante periodi di grande richiesta. Con l'IaC viene migliorata anche la sicurezza, infatti le organizzazioni possono definire delle policy di sicurezza e di configurazione rispetto all'infrastruttura. Queste policy possono essere controllate attraverso delle pipeline, ovvero delle catene di passaggi, che eseguono automaticamente dei test sul codice di configurazione dell'infrastruttura. Se questi codici violano le policy definite, allora il dispiegamento dell'infrastruttura fallisce automaticamente e vengono inviati dei feedback agli sviluppatori, in questo modo si evita la creazione di infrastrutture insicure [12]. Infine, l'IaC permette anche una riduzione sostanziale dei costi, sia perchè non è richiesto personale altamente specializzato per la configurazione manuale

e il dispiegamento delle risorse, sia perchè, essendo soprattutto un ambiente virtuale, viene diminuito il costo dell'acquisto di hardware proprietario, ad esempio, per server e database, poichè possono essere dispiegate molteplici risorse virtuali su un'unica risorsa fisica. Inoltre vengono ottimizzati i costi basati anche sul consumo del cloud [11], di fatti, le organizzazioni possono fornire risorse solamente quando necessario e non quando sono inattive.

Anche se molto utile ed efficiente, l'IaC contiene anch'esso alcune sfide e difficoltà nella sua adozione, specialmente all'inizio [1]. Infatti, bisogna considerare il cambiamento di adozione di strumenti da parte degli operatori IT, passando da operazioni manuali direttamente su risorse fisiche a alla definizione di codice che gestisce l'intera infrastruttura, che non è così immediato e talvolta anche complesso con l'utilizzo di specifici strumenti o software proprietari. Bisogna inoltre considerare le potenziali vulnerabilità di sicurezza che scaturiscono dalla definizione del codice per l'infrastruttura. Infatti, ad esempio, se in un file di configurazione viene definita una configurazione errata, ogni volta che quel file verrà applicato per creare l'infrastruttura, verrà applicato anche quell'errore di configurazione propagando così la vulnerabilità ad ogni provisioning. Talvolta, se non si definiscono delle chiare linee guida, i team di sviluppo potrebbero implementare l'IaC in modo incoerente. Inoltre, anche se effettivamente l'adozione di codice per definire un'infrastruttura in modo automatico diminuisce gli errori, rispetto a configurare le risorse manualmente ogni volta, è anche vero che se si presentano degli errori di configurazione nel codice è più complesso eseguire il debug. Per finire, l'adozione dell'IaC e la sua integrazione all'interno di un'infrastruttura già esistente non è una cosa semplice e può richiedere molto tempo, siccome bisogna modificare la precedente infrastruttura in modo da essere compatibile con il paradigma IaC.

2.3.4 Strumenti di Provisioning

Gli strumenti di provisioning sono proprio gli strumenti che vengono utilizzati per eseguire il processo di provisioning nel workflow dell'IaC descritto precedentemente. Questi strumenti si concentrano sull'impostazione iniziale e sulle configurazioni delle risorse dell'infrastruttura. Essenzialmente, gli strumenti leggono i file di configurazione e trasformano i requisiti definiti al suo interno nell'infrastruttura reale [11]. Di seguito sono elencati alcuni strumenti o servizi di provisioning.

AWS CloudFormation

È la soluzione nativa per l'Infrastructure as Code di AWS. Utilizza modelli come YAML o JSON per permettere l'istanziazione di risorse proprietarie AWS. Esso gestisce

automaticamente le dipendenze tra le risorse, creandole nell'ordine corretto e ripristinandole in caso di errori. CloudFormation permette una piena integrazione con tutti i servizi di AWS, mirata a semplificare il processo di definizione e di creazione di un'infrastruttura o applicazione. Tuttavia, questo strumento dichiarativo si interfaccia solo ed esclusivamente con i servizi e le risorse AWS, essendo infatti uno strumento di provisioning proprietario.

Azure Resource Manager

Similmente ad Amazon, Microsoft, con il suo servizio cloud Azure, ha implementato uno strumento di provisioning proprietario per gestire esclusivamente le risorse del proprio cloud. È anch'esso uno strumento di tipo dichiarativo basato su dei template scritti in JSON che offre alcuni vantaggi come il dispiegamento, la gestione e il monitoraggio di risorse Azure. Lo strumento utilizza un sistema integrato di "Role Based Access Control (RBAC)", ovvero un controllo di accesso basato sui ruoli, per gestire l'accesso di risorse e servizi e inoltre utilizza un sistema di tagging per l'organizzazione interna delle risorse.

Google Cloud Deployment Manager

Anche Google ha il suo strumento nativo proprietario utilizzato per gestire le risorse di Google Cloud Platform. È uno strumento dichiarativo che utilizza file di configurazione scritti in YAML per eseguire il provisioning delle risorse. Funziona sulla stessa linea d'onda degli strumenti automatizzati di Amazon e Microsoft.

Pulumi

Pulumi è una piattaforma di ingegneria cloud che permette di definire un'infrastruttura tramite software [13]. È molto differente dagli altri strumenti e piattaforme e utilizza un approccio unico all'IaC. Infatti, non definisce un linguaggio proprio ma piuttosto permette di scegliere linguaggi imperativi già esistenti, come TypeScript, Python, Go, C#, Java e YAML, insieme all'utilizzo di provider e librerie, tramite il suo "Software Development Kit (SDK)", per interfacciarsi con i servizi, permettendo così ai team di sviluppo l'utilizzo di linguaggi già conosciuti invece che imparare nuovi linguaggi dichiarativi proprietari di configurazione. Sebbene permetta l'utilizzo di linguaggi imperativi e i loro costrutti, come cicli, funzioni, blocchi condizionali e classi, il modello di descrizione dell'infrastruttura rimane dichiarativo, definendo quindi le risorse che devono essere presenti. Successivamente, il "deployment engine" interno di Pulumi gestisce il provisioning di ciò che viene definito nel programma di configurazione, funzionando quindi, a livello generale, nello stesso modo dell'engine interno di Terraform.

Terraform

Per finire, lo strumento di provisioning Terraform, sviluppato da HashiCorp, è uno strumento dichiarativo che permette di interagire con molteplici piattaforme cloud. Viene quindi scritto il codice nel linguaggio "HashiCorp Configuration Languague (HCL)" che si può interfacciare con diversi servizi cloud, inclusi quelli appena descritti, ma anche on-premise attraverso l'utilizzo di providers, che verranno illustrati nel dettaglio nella sezione successiva. In questo modo, Terraform aggira le dipendenze dei singoli servizi proprietari permettendo di interagire con più di uno contemporaneamente in base ai loro punti di forza o ai provider esistenti. Questo strumento è in grado di gestire le dipendenze tra risorse ma anche di crearle in parallelo, permettendo un provisioning molto veloce, limitato comunque dal servizio con cui comunica. Terraform è uno strumento ampiamente utilizzato per l'implementazione dell'IaC proprio grazie alla sua compatibilità con numerosissimi servizi esistenti.

Esistono anche altri strumenti automatizzati come Ansible, Chef, Puppet, che permettono la configurazione interna delle risorse, come ad esempio l'installazione di un software o un'applicazione, direttamente all'interno di una macchina virtuale. Questi strumenti vengono chiamati "Strumenti di Gestione della Configurazione" e sono utilizzati dopo aver eseguito il provisioning dell'infrastruttura, tuttavia non sono trattati all'interno di questa tesi.

2.3.5 Casi d'Uso e Procedure per IaC

L'utilizzo di codice per il provisioning di un'infrastruttura è molto utilizzato soprattutto nell'ambiente cloud, ma anche per facilitare varie operazioni IT di aziende e organizzazioni. In particolare, alcuni casi d'uso dell'IaC sono ad esempio il dispiegamento automatizzato delle applicazioni web, facilitano il dispiegamento delle risorse necessarie quali database, server, firewall e configurazioni di rete, definendole tutte in template riutilizzabili e riducendo il setup manuale, quindi, evitando errori di configurazione e diminuendo il tempo di provisioning. Altri casi d'uso possono essere l'integrazione multi-cloud, cioè la possibilità di utilizzare piattaforme cloud diverse contemporaneamente, sfruttando così gli specifici punti di forza di ogni piattaforma per determinate operazioni, un esempio di questo caso è, illustrato nella sottosezione precedente, lo strumento Terraform che permette l'interfacciamento con molteplici piattaforme contemporaneamente.

L'approccio IaC viene molto utilizzato anche nel paradigma di "Continuos Integration/Continuos Deployment (CI/CD)", un modello creato per accelerare il ciclo di sviluppo del software. Questa combinazione permette di automatizzare e integrare le modifiche al-

l'infrastruttura direttamente nell'ambiente di produzione e permette anche di mantenere altri ambienti, come quello di test e sviluppo, identici a quello di produzione facilitando così i test e aumentando l'affidabilità di sviluppo. Un altro aspetto importante dell'IaC è quello di automatizzare le configurazioni di sicurezza e le regole di conformità, o "compliance", infatti, tutte le configurazioni manuali dei criteri di sicurezza, come le definizioni di regole per i firewall, la gestione o definizione di ruoli, vengono pressochè eliminati, dal momento che essi vengono definiti come codice all'interno di file e applicati automaticamente all'interno dell'infrastruttura.

Infine, uno dei vantaggi più grandi che l'Infrastructure as Code fornisce è quello del "disaster recovery". Le organizzazioni mantengono varie versioni della definizione dell'infrastruttura in repository centrali e definisco dei piani di disaster recovery in caso di problemi o del fallimento dell'intera infrastruttura. Infatti, il provisioning automatico permette di rieseguire i file di configurazione in modo da riabilitare l'intera infrastruttura in tempi molto brevi.

Secondo "RedHat" [1], un approccio efficiente per l'IaC è quello di considerare l'infrastruttura come se fosse codice applicativo, e consiglia alcune procedure per applicare al meglio l'Infrastructure as Code:

Controllo delle Versioni - Utilizzare il controllo delle versioni per qualsiasi cosa all'interno dell'IaC favorisce lo sviluppo, tiene traccia delle varie versioni del codice e semplifica i rollback, facilitando così i disaster recovery come spiegato prima.

Infrastruttura Modulare - Rendere l'infrastruttura modulare significa dividerla nei suoi componenti più piccoli per poi riutilizzarli successivamente all'interno del codice. Indica che invece di scrivere la definizione di tutta l'infrastruttura all'interno di un singolo modulo, essa si divide in più moduli significativi dove, ad esempio, un modulo magari contiene le risorse della rete mentre un altro contiene quelle delle macchine virtuali. Successivamente, alla definizione vera e propria dell'infrastruttura si importano questi moduli invece di riscrivere il codice ogni volta sullo stesso singolo file, quindi questi moduli sono riutilizzabili. Questo aiuta anche a gestire le versioni.

Automatizzazione dei Test - Come già descritto, integrare l'IaC con test automatici in pipeline CI/CD permette che le modifiche apportate vengano controllate automaticamente e convalidate prima di eseguire il dispiegamento dell'infrastruttura.

Utilizzo di Infrastruttura Immutabile - Questo approccio aiuta a ridurre gli errori di configurazione, ridurre la coerenza tra ambienti diversi e semplificare i rollback,

poichè, invece che apportare le modifiche direttamente all'infrastruttura esistente in produzione, si punta ad eseguire il provisioning di una configurazione aggiornata e rimuovere quella precedente. Anche questo aiuta molto il versionamento dei file di configurazione.

2.4 Terraform come Tool di Provisioning per IaC

In questa sezione viene illustrato lo strumento di provisioning Terraform, proprietario di HashiCorp [14]. Terraform permette agli sviluppatori di eseguire il provisioning, l'aggiornamento e la rimozione di componenti di infrastrutture cloud o on-premise scrivendo dei file di configurazione in linguaggio dichiarativo. Uno sviluppatore definisce quindi lo stato desiderato delle risorse dell'infrastruttura, successivamente Terraform si occupa di creare automaticamente un piano di esecuzione, individuare le dipendenze definite tra risorse e creare i componenti nell'ordine corretto. Terraform può gestire componenti di basso livello come risorse di computazione, risorse di storage e di rete, così come componenti di alto livello, ad esempio configurazioni DNS e anche varie funzionalità di "Software as a Service (SaaS)".

2.4.1 Linguaggio di Configurazione

Il linguaggio di Terraform è dichiarativo e permette di definire come l'infrastruttura deve essere piuttosto che definire il modo in cui raggiungere tale stato [15]. Lo scopo principale del linguaggio è la definizione di risorse che rappresentano gli oggetti reali dell'infrastruttura, tutte le altre funzionalità sono presenti per rendere la definizione di risorse più scalabile e conveniente. La configurazione di un'infrastruttura può consistere in più file e cartelle all'interno del progetto Terraform.

La sintassi del linguaggio è molto semplice e consiste nel definire pochi specifici elementi:

Blocchi - I blocchi sono dei contenitori che rappresentano le configurazione di determinati oggetti, come le risorse. Un blocco è composto dal tipo del blocco, quindi una risorsa, un blocco dati oppure un blocco di output, una o più label, fornite dai provider per identificare i blocchi, e un corpo contenente vari argomenti ed eventualmente ulteriori blocchi interni.

Argomenti - Sono presenti all'interno dei blocchi. Essi hanno un nome a cui viene assegnato un valore.

Espressioni - Rappresentano dei valori, numerici o testuali, come stringhe, che vengono assegnati ad un argomento oppure ad altre espressioni

Di seguito un esempio concettuale di un blocco nel linguaggio Terraform:

```
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# Esempio reale
resource "aws_vpc" "main" {
  cidr_block = var.base_cidr_block
}
```

Nell'esempio sopra, viene mostrato come si crea una risorsa che rappresenta un "Virtual Private Cloud (VPC)" di AWS. Quindi il tipo di blocco è una risorsa, mentre le altre due label sono fornite dal provider, in questo caso sconosciuto, che permette di identificare un tipo specifico di risorsa. Infine, al suo interno è presente un argomento denominato "cidr_block" a cui viene assegnato un valore memorizzato in una variabile.

Nella definizione dei blocchi delle risorse l'ordine non conta, Terraform prende in considerazione solamente le dipendenze esplicite o implicite tra le risorse per determinarne l'ordine di creazione.

2.4.2 Componenti di Terraform

Il tool Terraform si affida a diversi componenti, interni ed esterni, per il suo funzionamento [16].

Moduli

Un modulo è una collezione di risorse che vengono utilizzate da Terraform come un unico blocco [17]. In pratica, un modulo è una directory contenente uno o più file ".tf" che vengono gestiti tutti insieme. I moduli sono utili soprattutto quando si hanno blocchi comuni che spesso devono essere riutilizzati. Ad esempio, se in uno o più file vengono definite delle risorse relative alle configurazioni di rete che vengono spesso riutilizzate, allora conviene inserirle in un modulo che può essere esportato e riutilizzato in altri progetti, in questo modo si riduce la duplicazione di codice.

Esistono due tipi principali di moduli:

Modulo Root - Il modulo root è la directory principale del progetto Terraform in cui vengono eseguite le operazioni di plan e apply. Solitamente è anche quello in cui vengono definiti i provider da utilizzare in tutto il progetto.

Modulo Child - I moduli child sono i "sottomoduli" che possono essere utilizzati per dare una struttura al progetto, ad esempio tutte le risorse di macchine virtuali possono essere inserite in un modulo mentre tutte quelle della rete vengono inserite in un modulo separato. Successivamente, quando è necessario utilizzarli, essi vengono richiamati dal modulo root e, di conseguenza, si può accedere alle definizioni dei loro blocchi interni per poterli utilizzare. Da notare che i moduli child possono avere anch'essi altri moduli child annidati, creando così una gerarchia di moduli.

I benefici principali dell'utilizzo di moduli sono l'organizzazione del codice, in questo modo si può raggruppare definizioni coerenti allo stesso ambito negli stessi moduli, il riutilizzo dei moduli, di fatto, piuttosto che ridefinire ogni volta risorse le stesse risorse, esse possono essere inserite in moduli e successivamente esportati per poterli riutilizzare. Infine, il loro utilizzo permette anche la condivisioni dei moduli stessi, permettendo ad altri utenti il loro utilizzo.

Risorse

Nel linguaggio Terraform, la definizione di una risorsa rappresenta un oggetto all'interno dell'infrastruttura da creare e gestire [18]. Una risorsa dipende esclusivamente dalla sua definizione da parte del provider, ad esempio il provider di AWS fornisce delle label e delle specifiche strutture per definire le risorse dei database e le macchine virtuali, così come Google Cloud e Microsoft hanno le loro proprie definizioni. La definizione di una risorsa, relativa alla struttura fornita dal provider, avviene definendo un blocco il cui tipo è appunto il tipo risorsa, dopodichè si definiscono le label, utilizzate per identificare la risorsa tra quelle che fornisce il provider, e successivamete si definiscono gli argomenti con i rispettivi valori. Alcuni argomenti possono essere obbligatori, altri opzionali, mentre altri ancora possono essere computati, cioè il loro valore viene fornito da Terraform a seguito del provisioning della risorsa, ad esempio l'id di una risorsa di rete può essere attribuito direttamente dal servizio cloud una volta che la risorsa viene creata, e successivamente restituito a Terraform. La definizione di una risorsa in un file, assieme a tutta la sua configurazione interna, rispecchia esattamente come l'oggetto corrispondente deve essere realmente all'interno del servizio cloud o on-premise. Infatti, al momento della creazione delle risorse, Terraform richiama le API necessarie del servizio

per poter comunicare le configurazioni delle risorse che i corrispettivi oggetti da creare devono possedere.

Provider

I provider sono dei plugin esterni, che vengono scaricati da Terraform e posti all'interno del progetto, che consentono a Terraform di interagire con dei servizi reali tramite l'utilizzo di API [19]. Senza i provider, Terraform è pressochè inutile, poichè non potendo comunicare con alcun servizio non sarebbe in grado di eseguire il provisioning delle infrastrutture. Un provider definisce un insieme di risorse e dati specifici per determinati servizi cloud o on-premise gestiti da Terraform e sa come comunicare con le API di tale servizio per poter creare l'effettivo oggetto reale nell'infrastruttura. I provider sono distribuiti separatamente da Terraform e ognuno di essi ha la propria versione e documentazione. La fonte principale è il registro di Terraform che contiene la maggior parte dei provider pubblici, sviluppati dalla community, da cui si possono scaricare tramite l'interfaccia testuale, da terminale, di Terraform eseguendo il comando "terraform init". In questo modo uno o più provider, definiti nel file "main.tf", vengono scaricati all'interno del progetto nel modulo root. Si può specificare direttamente la versione del provider da scaricare, altrimenti, non specificando alcuna versione, Terraform eseguirà il download dell'ultima versione disponibile. È anche possibile sviluppare i propri provider privati e integrarli successivamente in Terraform in locale.

Un esempio di provider viene fornito nella sezione 4.1.1 del capitolo 4.

State File

Terraform mantiene un file di stato che tiene traccia dell'infrastruttura che gestisce [20]. Questo "state file" viene memorizzato localmente e rappresenta tutto ciò che Terraform ha creato in base alle configurazioni fornite. Senza l'utilizzo dello state file, Terraform dovrebbe ogni volta eseguire una scansione per comprendere il contenuto di un'infrastruttura, ma questo procedimento sarebbe troppo lento. Quindi, Terraform utilizza lo stato per conoscere le risorse di cui ha già eseguito il provisioning, in questo modo può calcolare le differenze tra lo stato attuale e le modifiche definite che devono essere successivamente applicate. Lo stato mappa le risorse definite localmente con i propri corrispettivi reali già creati all'interno dell'infrastruttura e viene memorizzato all'interno del file "terraform.tfstate", creato automaticamente da Terraform e presente nel modulo root. Nel caso di un team di sviluppo, lo state file può anche essere memorizzato in modo condiviso tra tutto il team, offrendo così una visione a tutti gli sviluppatori dello stato attuale dell'infrastruttura, evitando conflitti e garantendo anche la coerenza tra tutto il team con le future modifiche.

Questo file è in formato JSON e contiene tutte le informazioni sugli stati delle risorse create, ad esempio gli id e i valori degli attributi di ogni risorsa. È fortemente sconsigliato modificare direttamente il file, poichè potrebbe generare incongruenze, e indicato invece l'utilizzo dell'interfaccia testuale di Terraform per gestirlo se necessario, ad esempio si può controllare l'elenco delle risorse nello state file con il comando "terraform state list" ed eventualmente mostrare i dettagli di ogni singola risorsa con il comando "terraform state show <nome_risorsa>". È sconsigliato modificare direttamente le risorse dispiegate sul servizio che si utilizza, quindi non tramite Terraform, in quanto anche questo procedimento porterebbe ad un'incongruenza tra lo stato effettivo dell'infrastruttura e lo stato mantenuto localmente da Terraform. Essenzialmente, lo state file è fondamentale per eseguire il plan e l'apply in modo coerente rispetto all'infrastruttura reale.

Terraform CLI

In conclusione, la "Command Line Interface" è l'interfaccia principale con cui interagire per gestire l'infrastruttura tramite Terraform [16], e viene utilizzata per dare istruzioni, tramite appositi comandi, a Terraform sulle azioni da compiere, come mostrare il piano di esecuzione e applicarlo, mostrare il contenuto dello state file, inizializzare l'ambiente Terraform con "terraform init" per scaricare i provider definiti e, tramite il comando "terraform init -upgrade", aggiornarli quando serve, oppure rimuovere completamente l'infrastruttura dal servizio ospitante con "terraform destroy".

2.4.3 Funzionamento di Terraform

Terraform può comunicare con qualunque piattaforma cloud o servizio attraverso l'utilizzo di API accessibili come ad esempio AWS, Azure di Microsoft, Google Cloud Platform, IBM Cloud e Docker [16]. Il ciclo di lavoro, o anche detto workflow, di Terraform per il provisioning di un'infrastruttura si divide in tre fasi: Write, Plan, Apply.

Write - Il procedimento iniziale è quello in cui viene definito il codice dichiarativo nel file di configurazione per definire le risorse da creare all'interno dell'infrastruttura. Si può definire un singolo file di configurazione che gestisce risorse presenti su molteplici piattaforme cloud o servizi differenti, utilizzando gli appositi provider.

Plan - Successivamente, per la creazione iniziale di un'infrastruttura, Terraform analizza il file di configurazione per determinare quali risorse deve creare utilizzando il comando "terraform plan". Alternativamente, se un'infrastruttura è già stata creata, Terraform confronta il file con il suo stato interno, memorizzato localmente, dell'infrastruttura attuale per vedere se sono state effettuate modifiche. Dopodichè, crea il piano di esecuzione che descrive come sarà l'infrastruttura finale,

esattamente come è stata definita nel file di configurazione. Il piano di esecuzione in sè è una lista che mostra quali risorse, e con quali configurazione, deve creare, in quale ordine, e nel caso di una modifica all'infrastruttura, mostra quali sono le differenze tra il piano di esecuzione appena creato e il suo stato interno della configurazione precedente. È importante notare che quando vengono eseguite delle modifiche alle risorse di un'infrastruttura già presente, se tali modifiche sono applicabili direttamente alla risorsa allora vengono semplicemente eseguite e applicate, ad esempio un tag o una descrizione. Se invece sono delle modifiche sostanziali e sono segnate dal provider che si utilizza come modifiche che richiedono la ricreazione della risorsa, allora Terraform procede all'eliminazione e nuovamente al provisioning della risorsa con le modifiche applicate. Essenzialmente, vuol dire che certe modifiche possiedono il concetto di immutabilità descritto nella sottosezione 2.3.2.

Apply - Infine, una volta che il piano di esecuzione è stato approvato, Terraform esegue le operazioni indicate nel piano tramite il comando "terrafom apply", quindi procede al provisioning dell'infrastruttura eseguendo le chiamate API necessarie alla creazione delle risorse nell'ordine corretto e rispettando le dipendenze tra esse, se presenti.

2.4.4 OpenTofu come Alternativa Open-Source

Terraform è stato creato nel 2014 da HashiCorp come software open-source sotto la licenza "Mozilla Public License (MSL)" e rimasto tale per nove anni [21]. Tuttavia, nell'agosto del 2023 HashiCorp ha deciso di cambiare la licenza di Terraform, passando alla "Business Source License (BSL)", ovvero una licenza considerata non open-source dalla "Open Source Initiative (OSI)". Questo cambio di licenza ha introdotto molta incertezza per le organizzazione che facevano affidamento su Terraform in quanto pone delle restrizioni sull'utilizzo commerciale del software, tuttavia non impone alcuna restrizione di utilizzo per i singoli utenti finali.

A seguito di questo evento, viene creata una "fork" dell'ultima versione Terraform sotto licenza MPL chiamata "OpenTofu". OpenTofu, governata dalla "Linux Foundation", viene creata appunto per contrastare questo cambio di strada da parte di HashiCorp, in modo da mantenere open-source tutta la comunità di sviluppo e di utilizzo del software Terraform e per evitare ulteriori cambi futuri di licenza potenzialmente anche più restrittivi. Di fatto, OpenTofu non ha alcuna differenza sostanziale al livello tecnico rispetto a Terraform, infatti, utilizza lo stesso linguaggio dichiarativo, gli stessi file .tf, la stessa gestione dello stato dell'infrastruttura, permette di definire dei moduli riutiliz-

zabili, supporta quasi tutti gli stessi provider, essendo molti di loro open-source. Una piccola differenza sviluppata da OpenTofu e attualmente mancante in Terraform è quella di adottare la crittografia dello state file dell'infrastruttura.

Essendo quindi entrambi i tool estremamente simili dal punto di vista tecnico e di sviluppo, la migrazione da progetti creati in Terraform verso il tool OpenTofu è molto semplice e veloce [22]. Infatti, nella maggior parte dei casi, basta installare la CLI di OpenTofu e successivamente, senza alcuna modifica nei file di configurazione e dei moduli, può essere utilizzata per eseguire il provisioning dell'infrastruttura definita precedentemente in Terraform, dal momento che anche i comandi principali quali init, plan e apply sono gli stessi anche nell'interfaccia di comando di OpenTofu.

Questo processo è stato testato anche nelle implementazioni presentate nel capitolo 4, sviluppate inizialmente in Terraform e successivamente testate anche in OpenTofu senza la necessità di alcuna modifica ai file di configurazione e di cambio di provider per Proxmox.

Capitolo 3

Software-Defined Networking e Infrastructure as Code per la Cybersecurity

La crescente adozione di paradigmi basati su software per la gestione delle infrastrutture IT, come il Software-Defined Networking e l'Infrastructure as Code, introduce importanti cambiamenti anche dal punto di vista della sicurezza. Da un lato, questi modelli permettono di migliorare il controllo, l'automazione e la consistenza delle configurazioni, contribuendo alla riduzione degli errori umani e alla possibilità di applicare meccanismi di sicurezza più velocemente e in modo centralizzato. Dall'altro lato, l'introduzione di nuove metodologie e componenti software comporta anche nuove superfici di attacco e vulnerabilità specifiche, legate proprio alla programmabilità dell'infrastruttura.

Questo capitolo si occupa quindi di analizzare i principali miglioramenti e le criticità di sicurezza introdotti da entrambi i paradigmi. In particolare, vengono prima esaminati i meccanismi di sicurezza e le vulnerabilità tipiche delle architetture SDN, per poi discutere dei benefici e dei rischi associati all'adozione dell'Infrastructure as Code per la gestione delle infrastrutture moderne definite tramite codice.

3.1 Miglioramenti e Vulnerabilità di Sicurezza di una SDN

L'adozione di una Software-Defined Network introduce nuovi aspetti di sicurezza che si dividono sia in benefici e miglioramenti ma anche nell'introduzione di nuove vulnerabilità, alcune specifiche proprio al modello di SDN. Tutto ciò è dovuto proprio da specifiche caratteristiche come la visione globale e centralizzata della rete grazie al controller e an-

che alla programmabilità della rete che permette di attuare modifiche e di implementare meccanismi di difesa automaticamente in tempo reale.

3.1.1 Controllo Dinamico del Flusso di Traffico

Il controllo dinamico del traffico all'interno della rete permette di implementare numerosi miglioramenti di sicurezza. In particolare, permette l'utilizzo di due tecniche principali [23].

La prima è il controllo dinamico degli accessi, che permette di monitorare il flusso di traffico tra tutti i nodi ed eventualmente cambiare delle regole in tempo reale. In una rete tradizionale, solitamente si utilizzano delle "Access Control Lists" (ACLs), posizionate all'interno di router oppure di "middlebox", come ad esempio dei firewall, per verificare se un determinato tipo di traffico è permesso. Ovviamente queste liste di controllo devono essere configurate e, se necessario, modificate manualmente su ogni router o firewall dedicato all'interno della rete. Inoltre, bisogna sottolineare che per mantenere una visibilità globale della rete, sarebbe richiesta la connessione di un firewall a tutti i nodi di rete tramite uno o più switch centrali, abilitando successivamente la copia di tutto il traffico da inoltrare verso il firewall, in modo da poter monitorare effettivamente tutto il traffico. Tuttavia questo è molto dispendioso in termini di gestione e configurazione e non sempre possibile.

In una SDN, invece, questi limiti vengono sorpassati. Infatti, il controller mantiene una vista centrale e globale di tutta la rete sottostante, permettendo così di interagire con tutti i nodi e di ottenere informazioni su tutto il traffico che transita tra di essi. Inoltre, non è richiesta l'installazione di middlebox addizionali, siccome è sufficiente l'utilizzo di uno switch o router, tipicamente funzionanti tramite il protocollo OpenFlow, che supportino le funzionalità di controllo degli accessi. In questo modo, il controller può comunicare direttamente con lo switch o router, e viceversa, e inviare modifiche di configurazione per le loro ACL interne, il tutto in modo dinamico e in tempo reale.

La seconda tecnica principale che il controllo dinamico della SDN permette è quello di individuare, ed eventualmente separare, il traffico malevolo da quello legittimo tra i diversi nodi di rete. Generalmente, il monitoraggio del traffico all'interno di una rete viene eseguito tramite l'utilizzo di strumenti come un "Network Intrusion Detection Sytem" (NIDS), che funziona anch'esso tramite l'utilizzo di regole e che esegue quella che viene chiamata "Deep Packet Inspection" (DPI), cioè non si limita, come nei firewall packet filters, a controllare solamente l'header dei pacchetti di rete, ma ispeziona anche il contenuto interno di essi, per verificare che questi non contengano ad esempio codice o comandi dannosi. Quando un NIDS rileva del traffico malevolo, invia solitamente un alert agli operatori di sicurezza specificando il tipo di traffico appena rilevato. Tuttavia,

a volte, si vuole investigare sul tipo di traffico malevolo che viene rilevato. Per fare ciò, vengono usati strumenti come un "HoneyPot". Questo strumento è un sistema o servizio volutamente vulnerabile posto all'interno della rete. L'idea è quella di esporre appositamente delle vulnerabilità per attirare un attaccante, facendogli credere di essere realmente entrato all'interno di un dispositivo critico della rete, monitorando le sue tecniche offensive e impedendo l'accesso ai sistemi effettivi, evitando o rallentando così una violazione di rete. In una SDN, questa tecnica può essere implementata controllando dinamicamente il traffico di rete al suo interno, cioè, secondo la tecnica appena illustrata, rilevando tramite un NIDS del traffico malevolo e reindirizzandolo dinamicamente verso un Honeypot, il tutto eseguito dal controller che modifica in tempo reale il flusso di traffico tra i nodi. Il paper [23] suggerisce due esempi relativi alle due metodologie appena mostrate.

Esempio del Firewall

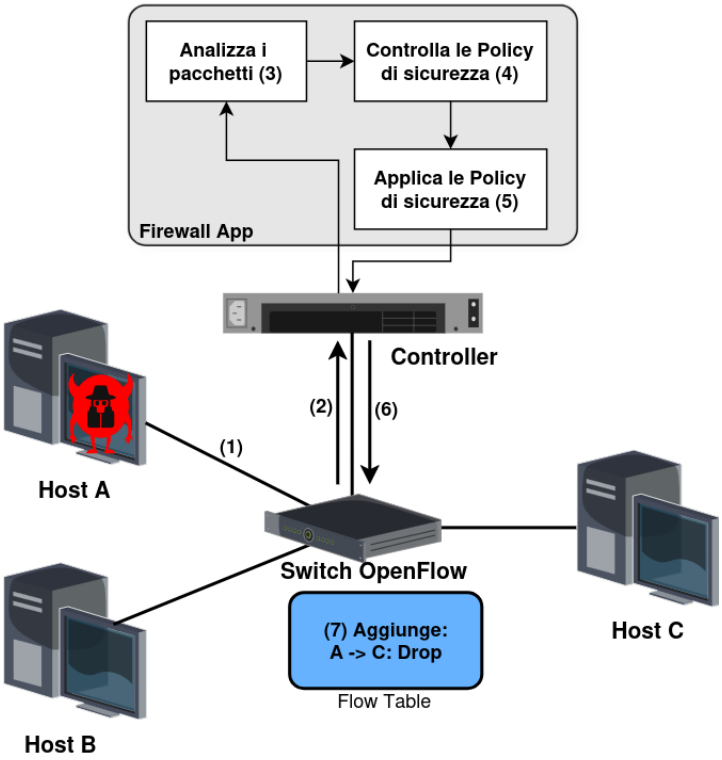

Figura 3.1: Esempio applicazione Firewall di alto livello

L'immagine 3.1 mostra la rappresentazione di una SDN minimale con il controller, tre host, uno switch con protocollo OpenFlow (OF) e un'applicazione di alto livello che comunica con il controller attraverso le interfacce Northbound, come descritto in 2.1.3.

Questo esempio mostra concettualmente ciò che è stato descritto in precedenza, ovvero la modifica della configurazione del firewall dello switch-OF eseguita dinamicamente dal controller insieme all'applicazione di alto livello, che appunto indica al controller cosa fare. In particolare, il traffico parte dal host A (1) dirigendosi verso lo switch-OF e successivamente, se lo switch non ha alcuna regola interna che determina cosa fare relativamente a quel tipo di traffico, lo inoltra al controller (2). L'applicazione, connessa al controller, riceve e analizza il traffico (3), in seguito controlla se questo tipo di traffico viola le policy di sicurezza (4), definite in precedenza, e definisce una regola per il firewall in base al riscontro con la policy (5). Quindi se il traffico è permesso allora il controller invia la nuova regola allo switch (6) che permette l'inoltro di quel tipo di traffico, se invece il traffico viene considerato come malevolo, o comunque non rispetta le policy, allora la regola inviata allo switch bloccherà quel tipo di traffico (7), impedendo l'inoltro verso altri host della rete.

Questo esempio tende a mostrare quanto discusso prima, cioè di come il controllo dinamico del traffico attuato dal controller e orchestrato eventualmente da applicazioni ad esso associate, possa definire il comportamento di un firewall dinamicamente e in tempo reale tramite delle semplici definizioni e applicazioni di regole. Ovviamente questo esempio concettuale può essere esteso anche ad altre casistiche e implementazioni di sicurezza, come l'utilizzo di un NIDS o anche di un "Network Intrusion Prevention System" (NIPS), che essenzialmente agisce come un rilevatore di traffico malevolo ma che è anche in grado di bloccarlo automaticamente, come host dedicato all'interno della rete che monitora tutto il traffico tra i nodi, comunicando direttamente con il controller.

Esempio del Honeypot

Come anticipato, a seguito del rilevamento di traffico malevolo si può agire in modo da bloccare totalmente quel tipo di traffico oppure lo si può reindirizzare verso un Honeypot per monitorarlo e studiarlo, così da sviluppare delle eventuali contromisure o policy di sicurezza per determinare le azioni da prendere in futuro nel caso quel tipo di traffico possa essere rilevato nuovamente. Questo esempio mostra come un traffico di rete può essere determinato come malevolo e reindirizzato a un Honeypot dinamicamente tramite il controller e un'applicazione specifica di alto livello. Sono presenti tre host di rete, di cui uno è l'attaccante e un altro è l'Honeypot, uno switch OpenFlow, il controller e l'applicazione per l'Honeypot. In questo esempio si assume che l'host A sia l'attaccante e che stia eseguendo un "port scanning", ovvero il processo di capire quali porte di un

sistema sono aperte, chiuse o filtrate, e quali servizi sono presenti dietro di esse, verso l'host B.

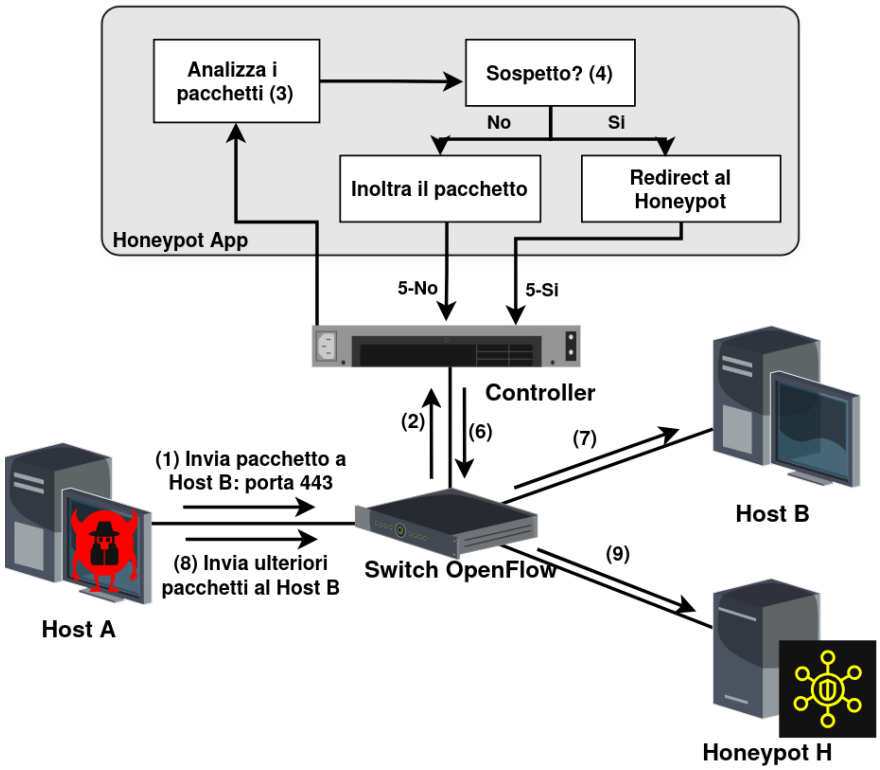

Figura 3.2: Esempio applicazione Honeypot intelligente

Inizialmente l'attaccante A invia una richiesta alla porta 443 destinata all'host B (1). Successivamente, lo switch-OF inoltra questo pacchetto al controller (2). L'applicazione per l'Honeypot riceve questo pacchetto, lo analizza (3) e in seguito determina se tale pacchetto è sospetto (4). Pertanto, se il pacchetto è legittimo, quindi non sospetto (5-No), come in questo caso, allora il controller comunica una nuova regola di instradamento allo switch (6) che permette il passaggio a quel tipo di traffico e quindi lo inoltra vero l'host B (7). Nel caso in cui il pacchetto sia sospetto (5-Si), allora il controller comunica

allo switch (6) la regola che instrada tale traffico verso l'Honeypot (9). Questo sarebbe il normale ciclo di esecuzione per il monitoraggio del traffico in questo esempio.

Successivamente l'attaccante continua l'invio di richieste verso diverse altre porte dell'host B (8). A questo punto, l'applicazione del Honeypot che comunica con il controller determina che l'host A sta scansionando l'host B, poichè rileva una grande quantità di richieste da parte dell'host A verso diverse porte dell'host B in brevissimo tempo, di conseguenza, installa immediatamente una nuova regola nello switch che inoltra tutto il traffico dell'host A verso l'Honeypot H. così facendo, l'attaccante A pensa di comunicare con l'host B mentre invece sta scansionando l'Honeypot.

Questo esempio mostra come, attraverso un'applicazione che può determinare un certo comportamento malevolo di un host della rete, in una SDN si possa agire automaticamente e in tempo reale per rilevare o, in questo caso, impedire comportamenti anomali e violazioni di rete. Naturalmente, l'esempio del port scanning è solamente uno dei tanti attacchi che possono essere effettuati verso gli host di una rete, lo stesso esempio potrebbe essere applicato ad un attacco "Distributed Denial of Service" (DDoS), anch'esso un attacco che può essere rilevato e bloccato o mitigato nella stessa maniera.

Tecnica del Moving Target Defense

Tipicamente, in una rete tradizionale la topologia è principalmente statica. Infatti, la maggior parte delle configurazioni dei sistemi e nodi di una rete, una volta eseguite, rimane invariata, come ad esempio indirizzi IP, percorsi di rete, determinati servizi offerti da diversi nodi di rete come dei webserver, ecc. Tuttavia, questa caratteristica di staticità rende una rete vulnerabile a molti attacchi, in particolare a quello dell'enumerazione. L'enumerazione è essenzialmente il processo di raccogliere più informazioni possibili rispetto ad una rete per costruire una mappa, in modo che un attaccante riesca a capire la sua topologia, i nodi presenti, gli indirizzi IP, quali sono i sistemi critici e soprattutto quali sono le vulnerabilità conosciute presenti all'interno della rete, una piccola parte dell'enumerazione è il port scanning descritto nell'esempio dell'honeypot in 3.1.1.

Per contrastare tali tipi di attacchi, viene introdotta la tecnica del "Moving Target Defense" (MTD) [24]. Essenzialmente, l'idea principale di questa tecnica è quella di riconfigurare costantemente una rete o sistema in modo da modificare dinamicamente la superficie di attacco. Così facendo, le informazioni raccolte da un attaccante in precedenza non sono più utili rispetto alla configurazione attuale del sistema, introducendo così maggiore complessità e tempo per un attaccante, limitando anche l'eventuale esposizione di vulnerabilità e incrementando la resilienza del sistema o della rete. Questa tecnica può essere considerata come una modalità di prevenzione per gli attacchi, introducendo imprevedibilità nella rete.

Le caratteristiche principali del modello SDN permettono una semplice e attuabile implementazione di questa tecnica. Infatti, la visione globale di rete insieme al controllo centralizzato e dinamico del flusso di traffico, permettono di modificare la topologia e il comportamento di rete agendo direttamente sul traffico e le configurazioni dei nodi. Sono state presentate, nel corso degli anni, due principali metodologie per attuare la tecnica MTD:

IP Mutation - Esistono varie tecniche per implementare la mutazione degli indirizzi IP. Una tecnica principale assegna due indirizzi IP a ciascun host di rete: quello reale, che non viene modificato, e quello virtuale, che viene aggiornato dal controller periodicamente e sconosciuto all'host stesso. Si assume una comunicazione interna tra due host H1 e H2. H1 invia un pacchetto di rete ad H2, in questo caso il pacchetto conterrà inizialmente l'IP sorgente reale di H1 e quello virtuale di H2. Successivamente, il pacchetto che arriva al primo switch OpenFlow, viene modificato sostituendo l'indirizzo reale di H1 con il suo virtuale e successivamente lo inoltra al secondo switch-OF. Il secondo switch, invece, si occupa di tradurre l'indirizzo di destinazione, da quello virtuale di H2 al suo reale. In questo modo la comunicazione avviene correttamente, tuttavia, H1 vedrà sempre l'IP virtuale di H2 e viceversa. Infatti, il lavoro principale viene eseguito dagli switch-OF che hanno il compito di tradurre correttamente gli indirizzi IP per ogni pacchetto di rete. Inoltre, gli indirizzi IP virtuali verranno cambiati periodicamente dal controller che comunicherà i nuovi indirizzi virtuali per ogni host agli switch OpenFlow. In questo modo, gli switch mantengono sempre la correlazione tra l'IP reale di un host e il suo IP virtuale, modificato periodicamente. Essenzialmente, questa tecnica si tratta di un "Network Address Translation" (NAT) dinamico e programmabile.

Route Mutation - La mutazione delle rotte di rete permette di modificare le rotte del traffico di rete da un host ad un'altro dinamicamente grazie al controller, che comunica periodicamente le nuove rotte agli switch OpenFlow, in modo far sembrare variabile la topologia di rete. Modificando le flow rules degli switch OpenFlow, il controller permette di variare nel tempo le rotte, assegnando a determinati host dei cammini alternativi per raggiungere altri nodi della rete, potenzialmente distribuendo anche il traffico su più percorsi. Esistono molte possibilità di implementazione per questa tecnica. In particolare, si possono modificare le rotte per far passare lo stesso flusso di traffico su nodi diversi dinamicamente, si può modificare la topologia di una rete attivando o disattivando i vari collegamenti logici tra nodi oppure selezionando percorsi alternativi periodicamente per ogni nodo. Questa tecnica viene considerata molto efficace per attacchi di tipo DDoS, in quanto permette

di modificare le rotte in modo da distribuire i vari tipi di traffico su i nodi della rete simultaneamente, così da diminuire il carico su ogni singolo nodo. Inoltre, un'altra possibilità è quella di modificare delle rotte per instradare un tipo di traffico, considerato malevolo o sospetto, verso i sistemi di sicurezza per il monitoraggio o come risposta ad una possibile violazione di rete. L'esempio del Honeypot, presente in 3.1.1, è appunto una dimostrazione di applicazione della tecnica Route Mutation.

Queste due tecniche sono state sviluppate in risposta ad attacchi noti di rete, in particolare IP Mutation nasconde l'indentità degli host di rete, mentre Route Mutation nasconde la struttura intera della rete ad un attaccante.

3.1.2 Controllo Centralizzato e Visione Globale della Rete

Avere una visibilità globale di una rete è molto importante come questione di sicurezza, infatti in tal modo è possibile monitorare e rilevare tutti gli eventi e i flussi di traffico presenti all'interno della rete tra i vari nodi. Esistono molti strumenti per eseguire il monitoraggio di rete, due dei quali già citati prima ovvero i NIDS e NIPS. Uno strumento utilizzato per mantenere una visione globale della rete è il "Security Information and Event Management" (SIEM) [25], ovvero un software in grado di ottenere varie informazioni da tutti gli host, o per lo meno quelli critici, della rete, correlando poi queste informazioni in modo tale da individuare eventuali comportamenti anomali o flussi di traffico malevoli all'interno della rete. Inizialmente, i sistemi SIEM funzionavano raccogliendo informazioni come i log da vari host della rete evolvendosi fino ad integrare sistemi di Machine Learning per l'individuazione di comportamenti anomali e minacce. Un altro strumento che viene utilizzato per il monitoraggio dei singoli sistemi è un "Endpoint Detection and Response" (EDR), un software installato direttamente all'interno di un host di rete che monitora i comportamenti e gli eventi che si verificano all'interno di un sistema registrandoli e memorizzandoli, utilizzare tecniche di analisi dati per rilevare comportamenti sospetti e bloccare attività o software malevoli all'interno di un sistema. Ovviamente un EDR monitora un singolo dispositivo pertanto necessità di essere installato su ogni sistema all'interno della rete.

Tutti questi strumenti e software che vengono utilizzati per il monitoraggio globale, necessitano specifiche configurazioni ed è anche complesso il loro dispiegamento in reti di grandi dimensioni. Tuttavia, il monitoraggio implementato in una SDN può essere molto più scalabile e veloce, facilitando di conseguenza i concetti di rilevamento e risposta alle minacce e violazioni della rete. Infatti, grazie alle caratteristiche principali di una rete programmabile come il controllo e la visione centrale e globale della rete, è possibile eseguire il monitoraggio degli host di rete molto più semplicemente comunicando con

essi per ottenere informazioni e statistiche relative al comportamento dell'intera rete. Le informazioni possono essere collezionate direttamente dal controller che comunica ed invia delle richieste agli host tramite l'utilizzo di un'applicazione di alto livello, così da analizzare i vari flussi di traffico e la topologia intera della rete.

Questa caratteristica di una SDN permette anche di migliorare l'utilizzo di applicativi di sicurezza, come quelli appena descritti. Normalmente in una rete tradizionale non è semplice o sempre possibile definire un'architettura o topologia di rete per permettere che tutto il traffico, o la maggior parte, venga monitorato da determinati dispositivi di sicurezza dispiegati all'interno della rete, questo perchè alcune connessioni possono essere complesse da installare a causa delle diverse posizioni fisiche o lontananze di alcuni dispositivi. Tramite una SDN, invece, il traffico presente all'interno della rete può essere modificato e reindirizzato per permettere che certi tipi di flussi passino direttamente per gli applicativi di sicurezza in modo da poter essere monitorati, esattamente come nell'esempio precedente dove il traffico malevolo viene rilevato e modificato per poter essere reindirizzato verso l'Honeypot.

Per fare un esempio che mostra come possono essere sfruttate queste capacità di una SDN, basatosi sull'esempio di [23], si potrebbe creare un'applicazione di alto livello che comunica con il controller e che permetterebbe di collezionare le informazioni dai vari host della rete, analizzarle, determinare se alcuni tipi di traffico sono malevoli o sospetti ed applicare delle nuove regole di instradamento per modificare tali flussi oppure installare delle regole di controllo degli accessi per bloccarli. In particolare, un software EDR è installato su tutti i dispositivi di rete, l'applicazione, tramite il controller, invia dei messaggi di richiesta ad ogni host per ottenere le informazioni sullo stato del sistema insieme a quelle del software EDR. Successivamente, l'applicazione riceve queste informazioni e le analizza. Se un qualche tipo di attacco o comportamento sospetto viene rilevato, allora il controller comunica delle nuove regole per bloccare quel tipo di traffico oppure modificarlo per instradarlo verso un applicativo di sicurezza come un NIDS o un Honeypot, come nell'esempio precedente. In base alle informazioni fornite dall'EDR relative ad uno specifico host di rete, se tale host viene considerato compromesso o infettato da malware, l'applicazione, ipoteticamente, potrebbe isolare completamente tale sistema, bloccando il traffico in ingresso e in uscita da esso, in modo da evitare la diffusione del malware verso altri host di rete.

Il punto dell'esempio appena mostrato è quello di sottolineare come tramite un'ipotetica applicazione si possano ottenere delle informazioni su tutti i nodi di rete, inviando delle semplici richieste, in modo da conoscerne gli eventi e lo stato generale. Questo è possibile proprio grazie alle caratteristiche della SDN come la visibilità globale e il controllo centrale della rete.

3.1.3 Programmabilità della Rete

Nelle reti tradizionali, gli applicativi di sicurezza, come quelli descritti precedentemente, sono tipicamente implementati e dispiegati su hardware apposito come nodi di rete che sono appunto i middlebox. Tuttavia, nell'architettura di una SDN, questi sistemi di sicurezza possono essere gestiti in due modi:

1. Come prima possibilità, gli applicativi di sicurezza possono essere integrati all'interno della SDN come veri e propri nodi di rete, che possono, di conseguenza, comunicare direttamente con il controller per mantenere un monitoraggio di rete sempre centralizzato. In questo modo, sistemi come i NIDS, NIPS o SIEM, possono essere gestiti tramite l'utilizzo di regole e policy di sicurezza che permettono la sorveglianza continua del traffico e la prevenzione di attacchi interni alla rete agendo direttamente sui flussi tra i vari host. Questa metodologia funziona esattamente come nell'esempio 3.1.1 in cui si possono avere, al posto di un Honeypot, i sistemi appena citati che comunicano direttamente con il controller, ed eventualmente un'applicazione di alto livello. Successivamente, quest'ultimo è in grado di applicare delle regole per poter manipolare un certo tipo di traffico e farlo passare attraverso un NIDS per il monitoraggio, oppure attraverso un NIPS per bloccarlo definitivamente.

2. La seconda possibilità è quella di implementare questi sistemi come delle applicazioni di alto livello. Invece che dispiegare direttamente dei sistemi di sicurezza all'interno della rete come hardware ad hoc, il concetto di questi sistemi viene implementato tramite la creazione di applicazioni, questo è proprio il concetto su cui si basa l'esempio 3.1.1. Infatti, nell'esempio del firewall è possibile notare come il concetto di quest'ultimo sia implementato all'interno di un'applicazione che riceve del traffico di rete, lo analizza ed infine, dopo aver determinato se tale traffico è sospetto o no, definisce delle regole che permettono il passaggio di tale traffico oppure lo bloccano. Queste regole, definite dall'applicazione e successivamente distribuite dal controller, vengono inviate allo switch OpenFlow che la aggiunge alla sua flow table. Questo modo facilita la possibilità di implementare tali metodologie di sicurezza senza l'utilizzo di hardware proprietario ad hoc che deve poi essere dispiegato fisicamente nella rete. Un'ulteriore esempio riguardo a questa funzionalità, verrà illustrato in 3.1.4, dove viene appunto mostrato come un NIDS può essere implementato tramite un'applicazione piuttosto che un sistema fisico.

3.1.4 Miglioramento di Sicurezza dei Sistemi e delle Informazioni

Tutte le funzionalità appena descritte di una SDN sono molto utili per migliorare l'efficienza e il dispiegamento di applicativi di sicurezza all'interno della rete che permettono il corretto monitoraggio ed evitano, per quanto possibile, le violazioni degli host presenti al suo interno. Queste caratteristiche specifiche migliorano i processi di prevenzione, rilevamento e risposta di incidenti all'interno della rete aumentando anche la sicurezza delle informazioni di una rete IT presenti in sistemi critici come i Database.

Prevenzione

Come suggerisce il nome, questo processo è molto importante per prevenire degli attacchi di rete impedendo agli attaccanti di eseguire determinate operazioni in modo da proteggere i nodi della rete e le informazioni che contengono, come appunto i database. Il processo di prevenzione, solitamente, viene implementato tramite la definizione e applicazione di determinate politiche di sicurezza che definiscono principalmente quale utente o sistema può eseguire che tipo di operazione accedendo a quale altro sistema. Questi principi fanno parte della triade di sicurezza "Authentication, Authorization and Accounting" (AAA) [26]:

Authentication - Il processo di Autenticazione coinvolge un utente nel fornire delle informazioni che determinano "chi è l'utente". Solitamente il processo viene eseguito tramite un login in cui l'utente fornisce lo username o email e la password. Successivamente il server che si occupa del controllo del login, confronta le informazioni fornite con un database interno per verificare se esiste una coppia username, o email, e password fornita nel login. Esistono principalmente tre tipi di autenticazione che riguardano ciò che un utente sa, esattamente come una password nell'esempio appena illustrato, ciò che un utente possiede, come ad esempio un token di autenticazione pre-generato, oppure ciò che un utente è, ovvero l'impronta digitale o qualsiasi altro metodo biometrico di riconoscimento.

Authorization - In seguito all'autenticazione, è presente il processo di Autorizzazione, che determina "cosa l'utente può fare". Sostanzialmente, dopo che il sistema riconosce chi è l'utente, determina i privilegi che esso possiede che si traducono in ciò che l'utente può fare come ad esempio accedere a determinati sistemi, leggere o modificare determinati file oppure accedere a determinate zone di una rete fino ad intere sottoreti. Esistono diversi tipi di definizione dei privilegi come l'assegnamento diretto di essi ad un utente, oppure dei modelli come il "Role Based Access Control" (RBAC), ovvero un modello che definisce dei ruoli a cui sono associati dei

privilegi, e successivamente assegna tali ruoli a degli utenti. In questo modo un utente riceverà automaticamente i privilegi associati al suo ruolo.

Accounting - Infine, il "Tracciamento", o "Auditing", è il processo di tenere traccia, cioè memorizzare, tutte le attività di un utente sui sistemi di una rete interna. Viene spesso implementato tramite l'utilizzo di log che determinano cosa l'utente ha fatto, in quale momento e su quale sistema o risorsa. Per fare un esempio, se un utente esegue una query per interrogare il database in modo da ottenere specifiche informazioni, il sistema crea un log che indica a quale sistema l'utente ha avuto accesso, il tempo, ovvero la data e l'ora, in cui l'accesso è avvenuto e potenzialmente anche da quale sistema è stato eseguito l'accesso. Con queste informazioni è possibile ricostruire le attività di un utente all'interno della rete. Questo processo è fondamentale in quanto si occupa di monitorare costantemente gli utilizzi di determinati sistemi da parte degli utenti per verificare che i privilegi vengano rispettati ed, eventualmente, in caso della compromissione di un account utente, controllare le attività malevole di tale account per monitorare quali sistemi o file potrebbero essere stati compromessi in modo da attuare le dovute contromisure.

Queste politiche vengono applicate come misure di sicurezza proprio per garantire il corretto utilizzo di determinati sistemi critici, come appunto l'accesso a un Database o un server. Si possono anche definire delle politiche in grado di garantire il principio di minimo privilegio, ovvero un concetto secondo cui ad un utente o processo devono essere assegnati solo ed esclusivamente i privilegi, anche elevati, per eseguire solamente le operazioni richieste per il tempo necessario, e non oltre. Proprio per l'importanza che queste politiche di sicurezza rappresentano, esse devono essere definite in modo preciso e meticoloso per evitare errori di configurazione che possono poi portare a bloccare un utente legittimo oppure ad accettare un comportamento dannoso o malevolo.

In una SDN, questo processo viene attuato tramite la funzionalità del controllo dinamico del flusso implementando appunto funzioni di controllo degli accessi in modo automatico e dinamico. Come descritto nell'esempio in 3.1.1, queste policy possono essere applicate o modificate semplicemente aggiungendo o rimuovendo delle regole nelle tabelle degli switch OpenFlow, o in altri dispositivi di rete che utilizzano il protocollo OpenFlow, in tempo reale. Tuttavia, anche se questa funzionalità migliora il processo di prevezione, potrebbe potenzialmente far sorgere un nuovo problema chiamato "dynamic flow tunneling". È possibile che il controllo del flusso dinamico permetta a del traffico malevolo di evadere le politiche di controllo degli accessi.

Rilevamento

Il processo di rilevamento si occupa essenzialmente di monitorare il traffico della rete per controllare che non siano presenti anomalie o comportamenti malevoli che potrebbero potenzialmente indicare la presenza di attacchi di rete. In particolare vengono identificati il rilevamento degli abusi (misuse detection), che si tratta di rilevare degli attacchi di rete basati su pattern noti, cioè le "signatures" del traffico o degli attacchi, e il rilevamento delle anomalie (anomaly detection), che essenzialmente controlla che non siano presenti dei comportamenti sospetti da parte di utenti o sistemi [23]. La funzionalità di controllo centralizzato e visione globale della rete di una SDN permette il miglioramento delle prestazioni per il rilevamento delle anomalie. Questa funzionalità permette a un sistema di rilevamento di avere una visibilità dell'intera rete in modo da monitorare tutti i nodi connessi, permettendo così la raccolta di molte più informazioni utili per l'analisi. Inoltre, è utile anche per il rilevamento degli abusi, infatti si potrebbe programmare il data plane della SDN in modo da inoltrare tutto il traffico di rete direttamente verso il controller, grazie alla funzionalità del controllo centralizzato, che potrebbe utilizzare delle applicazioni esterne per eseguire una ispezione profonda dei pacchetti (DPI), per analizzare i loro contenuti. Riguardo a quest'ultima premessa, viene mostrato un esempio, fornito dal paper [23], che suggerisce proprio un'ipotetica implementazione per il rilevamento degli abusi e delle anomalie, secondo le funzionalità della SDN appena descritte.

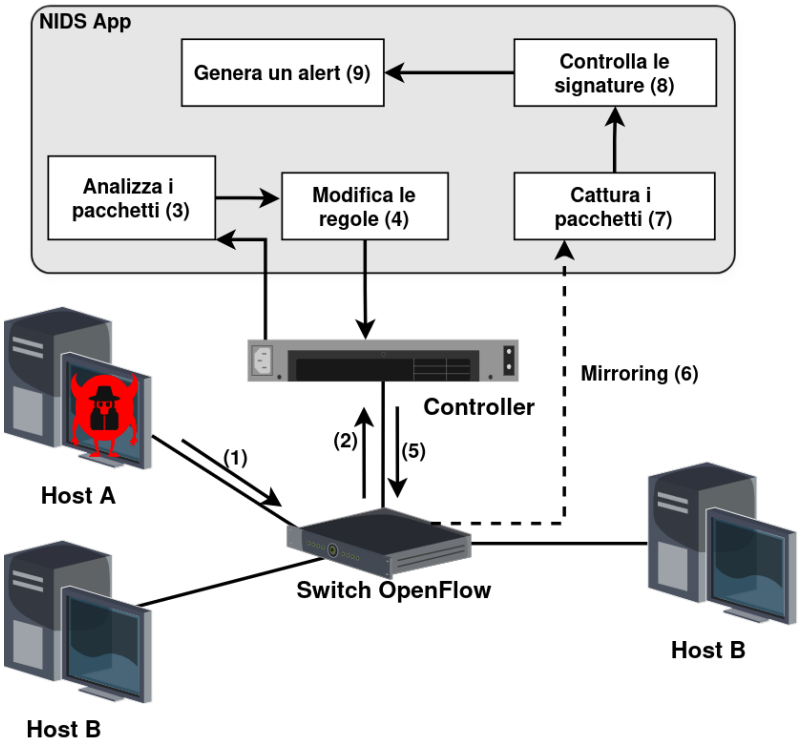

Figura 3.3: Esempio applicazione Network Intrusion Detection System

Un host A, considerato come attaccante, invia dei pacchetti di rete verso altri host (1) e, siccome tale traffico di rete risulta essere un nuovo tipo di traffico, lo switch OpenFlow non contiene alcuna regola nella sua flow table che gli permetta di maneggiare correttamente questo tipo di traffico, pertanto inoltra tale traffico verso il controller (2). A questo punto, l'applicazione che comunica con il controller, che rappresenta un NIDS a livello software, riceve i pacchetti di rete e li ispeziona per verificare se quel tipo di traffico presenta della anomalie di sicurezza oppure rappresenta un comportamento malevolo (3). In seguito, se il NIDS segnala quel tipo di traffico come sospetto, definisce delle nuove regole (4) che il controller comunica allo swtich-OF (5). Queste regole indicano allo switch di eseguire il mirroring del traffico sospetto, ovvero che tale flusso di traffico deve essere inoltrato alla sua destinazione come previsto, ma anche copiato e inoltrato al controller

(6), che a sua volta lo invia all'applicazione. Infine, l'applicazione NIDS riceve il traffico sospetto copiato (7) ed esegue una DPI per determinare se tale traffico è effettivamente malevolo, controllando che i pacchetti di rete corrispondano a signatures conosciute (8) e, se dei pacchetti corrispondono, allora il NIDS invia degli alert (9) indicando che è stato rilevato del traffico di rete malevolo.

Ovviamente, anche questo esempio può essere riprodotto utilizzando un NIPS che, invece di limitarsi a generare ed inviare degli alerts, è capace di bloccare preventivamente il traffico malevolo, grazie anche all'applicazione in tempo reale di regole di sicurezza.

Questo esempio mostra come è possibile implementare un sistema di rilevamento che cattura tutto il traffico di rete e lo convoglia in un punto centrale grazie alle caratteristiche del controllo centralizzato e della visione globale di una SDN. Inoltre, le funzionalità di sicurezza, come il NIDS o NIPS, possono essere facilmente implementate non come sistemi posizionati su hardware all'interno della rete ma come applicazioni di alto livello che comunicano con il controller, il tutto grazie alla programmabilità che una SDN è in grado di offrire.

Risposta

Il processo di risposta ad attacchi di rete viene attuato solamente dopo che questi si sono verificati. Solitamente, per mitigare un attacco si necessita dell'installazione di middlebox all'interno della rete che sono in grado di scartare (DROP) o rifiutare (REJECT) i pacchetti di rete del traffico malevolo, come ad esempio i NIPS introdotti in precedenza, ed eventualmente isolare, quindi mettere in quarantena, gli host infetti per evitare il diffondersi di malware o ulteriori violazioni ad altri sistemi.

In una SDN, il processo di risposta può essere migliorato ancora una volta dalla funzionalità del controllo dinamico del flusso di traffico che permette di definire delle regole per bloccare certi tipi di traffico considerati malevoli o sospetti, come negli esempi del firewall e dell'Honeypot, in cui viene mostrato come possono essere applicate delle regole in modo dinamico subito dopo il rilevamento di traffico malevolo all'interno della rete per bloccare tale traffico o reindirizzarlo verso un applicativo di sicurezza. Inoltre, questa funzionalità permette anche di applicare delle regole che isolano completamente un host di rete, considerato compromesso, per applicare appunto la quarantena a tale host. Infine, la programmabilità della rete permette di creare delle applicazioni di alto livello che fungono esattamente da applicativi di sicurezza, proprio come negli esempi illustrati finora in cui queste applicazioni possono rappresentare le funzionalità di sicurezza di un NIDS, di un firewall oppure per l'implementazione di un Honeypot automatico e intelligente.

3.1.5 Micro-Segmentazione in una SDN

Tradizionalmente, la segmentazione di rete, chiamata anche macro-segmentazione, definisce delle politiche di sicurezza che regolano il traffico tra domini di rete relativamente ampi, come ad esempio la comunicazione tra due intere subnet, consentendo, tuttavia, comunicazioni piuttosto libere all'interno della stessa zona, cioè una connessione di rete non viene sottoposta a politiche o controlli di sicurezza, da parte ad esempio di firewall, se avviene, tipicamente, tra due host della stessa zona [27].

La micro-segmentazione di rete, invece, rappresenta un'evoluzione del concetto di segmentazione tradizionale, in cui l'isolamento di rete non avviene più eclusivamente tra sottoreti o zone separate a livello logico, ad esempio le VLAN, ma viene applicato direttamente a livello dei singoli nodi di rete o servizi che essi espongono, come un database o un webserver.

La micro-segmentazione, quindi, introduce un modello di controllo molto più selettivo, in cui ogni host di rete può comunicare esclusivamente con altri host esplicitamente autorizzati e solo per specifici tipi di traffico. Questo approccio riduce significativamente la superficie di attacco e limita il movimento laterale di un attaccante all'interno della rete [28], ciò significa che una volta violato un nodo della rete è estremamente limitata, tuttavia non impossibile, la possibilità di violare o infettare altri nodi interni poiché sono attive delle regole che permettono solo ed esclusivamente specifiche connessioni tra gli host. Tradizionalmente, la micro-segmentazione è implementata tramite firewall distribuiti sugli host oppure anche su nodi di rete ad-hoc come i firewall perimetrali dei router. Tuttavia tali soluzioni risultano a volte complesse da configurare e mantenere, in quanto si basano su indirizzi IP statici e sulle configurazioni definite sui singoli dispositivi di rete.

L'architettura SDN migliora e semplifica la micro-segmentazione grazie alla visione globale e al controllo centralizzato e dinamico del flusso di traffico. Il controller, può infatti definire e distribuire dinamicamente regole di comunicazione tra i singoli host della rete, aggiornandole automaticamente in tempo reale. Ad esempio, il controller può installare delle regole negli switch OpenFlow che permettono il passaggio del traffico specificato e bloccano tutto il resto, come già visto negli esempi precedenti, oppure può comunicare le regole di filtraggio direttamente ai singoli host, in modo che ogni sistema della rete sia isolato e che possa comunicare con gli altri nodi solo ed esclusivamente tramite il traffico specificato dalle regole. La SDN consente di implementare in modo scalabile modelli di sicurezza di tipo Zero Trust, cioè negando tutte le comunicazione eccetto quelle esplicitamente autorizzate, migliorando la capacità di prevenire e mitigare attacchi interni attraverso l'isolamento dinamico e automatico dei nodi della rete.

3.1.6 Vulnerabilità Note di una SDN

Basandosi su quanto illustrato nella Sezione 2.1, una Software-Defined Network si basa su un modello molto differente dalle reti tradizionali. Tuttavia, se ciò permette di introdurre nuove tecniche e capacità per migliorare la sicurezza di una rete, questo modello presenta anche nuove vulnerabilità specifiche, introducendo nuovi punti critici che, di conseguenza, modificano la superficie di attacco di una rete. In particolare, componenti come il controller, il protocollo OpenFlow e le API, che rappresentano rispettivamente il controllo centralizzato, il controllo dinamico del flusso e la programmabilità della rete, rappresentano, per un attaccante, obiettivi di grande importanza e valore che, se violati, possono portare alla compromissione dell'intera rete.

Sebbene alcuni vettori di attacco sono comuni alle reti tradizionali, altri sono molto più specifici al modello SDN e soprattutto indipendenti dalle tecnologie e protocolli utilizzati, poichè rappresentano delle minacce rispetto al modello concettuale e architetturale di una SDN [29].

Le principali vulnerabilità possono essere analizzate considerando i tre elementi fondamentali dell'architettura SDN: il controller, il protocollo per il data plane, tipicamente OpenFlow, e le API northbound insieme alle applicazioni di alto livello.

Vulnerabilità e Attacchi al Controller

Il controller rappresenta il componente più critico dell'architettura SDN, in quanto possiede una visione globale della rete e la capacità di comunicare regole di inoltro nei dispositivi sottostanti. Di conseguenza, la sua compromissione può comportare il controllo completo dell'infrastruttura di rete.

Una prima categoria di vulnerabilità riguarda l'accesso non autorizzato al controller. Poiché il controller espone interfacce di gestione e API di controllo, meccanismi di autenticazione deboli o configurazioni errate possono consentire ad attaccanti di ottenere privilegi elevati. In tal caso, un attaccante può modificare le politiche di rete, inserire regole di inoltro malevole o intercettare traffico. In particolare, se un attaccante ha successo nella compromissione del controller potrebbe creare delle regole malevole da comunicare agli switch OpenFlow per compromettere il traffico dell'intera rete. Generalmente, l'effetto di tale attacco comporterebbe a un "Man In The Middle" (MITM), ovvero l'attaccante sarebbe in grado di intercettare tutto il traffico, manipolarlo o bloccarlo completamente. Per fare alcuni esempi, l'attaccante potrebbe reindirizzare tutto il traffico verso dei nodi che egli controlla, che porterebbe potenzialmente ad un'esfiltrazione di dati, potrebbe installare tali regole per disattivare completamente i sistemi di sicurezza dispiegati nella rete oppure potrebbe creare dei canali di comunicazione nasco-

sti per mantenere una persistenza all'interno della rete. Infine, un attaccante portebbe modificare le regole già presenti negli switch, oppure eliminarle direttamente, per bloccare determinati nodi e servizi di rete che porterebbe dunque ad un attacco Denial of Service (DoS).

Un'altra problematica critica è rappresentata dagli attacchi DoS al controller. Poiché il controller gestisce le richieste provenienti dagli switch, ad esempio per nuovi flussi sconosciuti oppure per flussi di traffico da monitorare come nell'esempio in 3.1.4, un attaccante può generare grandi quantità di traffico o di richieste di installazione di flussi, saturando le risorse del controller. Data la sua centralizzazione, tale attacco al controller può compromettere l'intera rete.

Un ulteriore vettore di attacco è la manipolazione della topologia e dello stato di rete. Il controller mantiene la propria visione della rete in base alle informazioni ricevute dagli switch, perciò, se tali informazioni vengono falsificate o manipolate, il controller può eseguire azioni errate, come la comunicazione di regole inesatte agli switch che potrebbe comportare la gestione sbagliata di alcuni flussi di traffico di rete. Ciò può portare a instradamenti errati o non ottimali, la deviazione del traffico o la disabilitazione involontaria di alcuni percorsi, il chè porterebbe in certi casi all'isolamento di alcuni nodi della rete.

Tra le soluzioni generali proposte nel corso degli anni, e presentate in [30] e in [29], vengono elencate ad esempio l'uso di "failover" per il controller, ovvero l'impiego di controller ridondanti in modo da eliminare il "Single Point of Failure" che essi rappresentano, meccanismi di autenticazione forte e gestione dei privilegi, monitoraggio delle risorse del controller e sistemi di rilevamento delle anomalie nel piano di controllo e nel piano dati.

Vulnerabilità del protocollo OpenFlow e del Data Plane

Il protocollo OpenFlow costituisce il principale meccanismo di comunicazione tra il controller e gli switch in una SDN. Attraverso tale protocollo, il controller installa, modifica e rimuove le flow rules nei dispositivi di rete. La sicurezza delle comunicazioni tramite le interfacce Southbound, ovvero tra il controller e gli switch, è quindi essenziale per garantire l'integrità e l'autenticità del controllo della rete.

Una vulnerabilità fondamentale riguarda la mancata autenticazione o protezione del canale di comunicazione tra il controller e gli switch. In alcune implementazioni, la comunicazione OpenFlow può avvenire senza cifratura o con autenticazione debole. In particolare, l'utilizzo del protocollo TLS per tali comunicazioni, è consigliato ma non obbligatorio e implementato a prescindere. Ciò consentirebbe attacchi di tipo MITM, come esplicato per il controller, che porterebbero all'intercettazione o manipolazione delle flow rules comunicate dal controller agli switch. Un attaccante che intercetti o modifichi

messaggi OpenFlow può installare regole arbitrarie o modificare quelle già esistenti negli switch per manipolare il comportamento dell'intera rete.

Un'altra minaccia rilevante è lo "spoofing" del controller o dello switch. Se un dispositivo malevolo riesce a impersonare il controller, può inviare comandi agli switch e ottenere il controllo del traffico. Analogamente, uno switch compromesso può inviare informazioni false al controller, manipolando la percezione della topologia o dello stato di tutto il traffico di rete.

Le SDN sono inoltre vulnerabili ad attacchi di saturazione delle flow table degli switch (flow table exhaustion). Gli switch OpenFlow dispongono di memoria limitata per memorizzare le regole di inoltro. Pertanto, un attaccante può generare un elevato numero di nuovi flussi, costringendo lo switch a richiedere continuamente decisioni al controller e riempiendo le tabelle. Ciò provoca una degradazione delle prestazioni, l'aumento della latenza e una potenziale perdita di traffico legittimo tra i nodi.

Infine, un ulteriore problema riguarda la dipendenza dal controller per i flussi sconosciuti. In una SDN, ogni nuovo flusso viene inviato dagli switch al controller per determinare la decisione da prendere, gli switch richiedono al controller delle regole per maneggiare tale tipo di traffico. A questo punto, un attaccante può sfruttare questo comportamento generando del traffico diverso frequentemente, causando un carico eccessivo sul canale southbound e sul controller stesso, portando così ad attacchi di tipo DoS, come illustrato in precedenza per il controller, e di "flow table exhautsion" per gli switch.

Tra le contromisure generali proposte sono presenti l'uso obbligatorio del protocollo TLS per il canale OpenFlow, meccanismi di autenticazione reciproca tra controller e switch, limitazione del numero di richieste di flusso, politiche di installazione proattiva delle regole, ovvero la preinstallazione delle regole negli switch per flussi di traffico comuni in modo da poterli maneggiare immediatamente senza l'invio di richieste al controller, ed infine tecniche di rilevamento di traffico anomalo o di saturazione delle tabelle, quindi il rilevamento di traffico malevolo che può comportare ad attacchi DoS o flow table exahustion.

Vulnerabilità delle API Northbound e delle Applicazioni di Alto Livello

Le API northbound rappresentano l'interfaccia attraverso cui applicazioni di alto livello interagiscono con il controller di una SDN. Esse consentono di programmare il comportamento della rete a livello logico, implementando delle funzionalità come il routing, la sicurezza, la qualità del servizio (QoS) e il monitoraggio. Tuttavia, l'esposizione di tali API introduce significativi rischi di sicurezza, soprattutto se tali interfacce funzionano senza i dovuti meccanismi di autenticazione.

Una delle principali problematiche, infatti, è l'accesso non autorizzato alle API northbound. Meccanismi di autenticazione insufficienti o gestione inadeguata delle credenziali possono consentire a utenti o applicazioni non autorizzate, e possibilmente malevole, di inviare comandi al controller. Poiché le API consentono la modifica diretta delle policy di rete, l'accesso illegittimo equivale di fatto al controllo dell'intero traffico della rete sottostante. Inoltre, le applicazioni, che comunicano con il controller tramite le API Northbound, possono operare con privilegi elevati e accedere alle funzioni di configurazione della rete. Se tali applicazioni sono vulnerabili o malevole, esse possono installare regole di inoltro arbitrarie, deviare traffico, disabilitare meccanismi di sicurezza o creare canali nascosti di comunicazione. La compromissione di una singola applicazione può quindi propagarsi all'intera infrastruttura.

Le API northbound, essendo molto spesso implementate come API REST, introducono anche vulnerabilità tipiche delle applicazioni web, quali injection di comandi, manipolazione dei parametri o accesso non autorizzato a risorse interne del controller. Tali vulnerabilità possono essere sfruttate per ottenere privilegi elevati o eseguire comandi di rete non autorizzati.

Vengono proposte varie soluzione generiche tra cui rientrano i modelli di controllo degli accessi basati su ruoli o capacità, autenticazione forte e gestione sicura delle credenziali API, verifica e certificazione delle applicazioni di alto livello di terze parti, nonché il monitoraggio del comportamento di tali applicazioni e il rilevamento di anomalie nelle richieste northbound.

3.2 Miglioramenti e Vulnerabilità di Sicurezza introdotte dall'IaC

L'adozione del paradigma IaC non comporta soltanto vantaggi operativi e di scalabilità, in termini di creazione e configurazioni di risorse, ma produce effetti rilevanti anche sulla sicurezza delle infrastrutture, in quanto consente di ridurre errori di configurazione manuali da parte di tecnici e operatori, aumentare la tracciabilità delle modifiche, tramite l'utilizzo di versioni per i template, e integrare controlli di sicurezza nel ciclo di vita dell'infrastruttura, specialmente durante lo sviluppo del codice che la definisce. In generale, l'adozione dell'IaC, anche nel mondo cloud, contribuisce a rafforzare la postura di sicurezza complessiva dei sistemi, utilizzando appunto la standardizzazione delle configurazioni, l'automazione dei controlli durante lo sviluppo e la verificabilità e il versionamento delle modifiche all'infrastruttura.

3.2.1 Consistenza delle Configurazioni

Uno dei principali miglioramenti di sicurezza dell'IaC è la possibilità di definire configurazioni infrastrutturali sicure tramite definizione di tipo dichiarativo e riutilizzabile, permettendo la creazione di ambienti coerenti e riproducibili. La definizione dell'infrastruttura come codice consente infatti di applicare degli standard di sicurezza uniformi, definendo anche come il codice deve essere scritto, ad esempio evitando la scrittura di credenziali di accesso a servizi cloud all'interno di esso, ed evitando differenze tra ambienti dovute a configurazioni manuali o modifiche dirette ai sistemi. L'utilizzo di template e moduli riutilizzabili consente di definire configurazioni già validate dal punto di vista della sicurezza, che possono essere propagate automaticamente nell'infrastruttura [31]. In questo modo, configurazioni ritenute sicure possono essere definite una sola volta e riapplicate in modo sistematico, riducendo la possibilità di errori di configurazione e inconsistenze dovute dall'applicazione di modifiche manuali.

Un ulteriore beneficio significativo riguarda la riduzione degli errori umani nelle attività di configurazione dell'infrastruttura. Le configurazioni manuali tradizionali rappresentano una delle principali fonti di vulnerabilità, poiché tali configurazioni possono essere soggette a mancanze, errori di sintassi, errori nell'assegnemento di permessi, potenzialmente permessi eccessivi ad utenti di livello non elevato, o esposizioni accidentali di servizi non permessi o inutilizzati. L'IaC sostituisce gran parte delle operazioni manuali con processi automatizzati nei quali l'infrastruttura viene generata a partire da specifiche definite tramite codice e verificabili. L'automazione e la definizione delle configurazioni tramite codice consentono quindi di ridurre le misconfigurazioni e di garantire maggiore consistenza tra gli ambienti di sviluppo, di test e di produzione [32].

3.2.2 Automazione e Integrazione della Sicurezza nello Sviluppo di Codice Infrastrutturale

L'IaC introduce, inoltre, la proprietà di versionamento e tracciabilità delle modifiche eseguite all'infrastruttura. Poiché le configurazioni sono gestite come codice, esse possono essere memorizzate in sistemi di controllo delle versioni, consentendo di mantenere una lista completa delle modifiche eseguite tramite l'utilizzo di versioni, identificare gli autori delle modifiche e ripristinare rapidamente configurazioni precedenti in caso di errore o compromissione, quindi eseguire un rollback immediato e sicuro. Il versionamento dell'infrastruttura permette inoltre i processi di audit, cioè la memorizzazione di tutte le azioni ed eventi avvenuti tramite la registrazione di log, e revisione analoghi a quelli del software, migliorando la gestione delle modifiche e la sicurezza di quest'ultime [31]. Questo approccio consente di applicare dei meccanismi di revisione del codice anche alle modifiche nei template di un'infrastruttura, essendo essi definiti tramite codice, introducendo controlli preventivi, prima ancora del provisioning dell'infrastruttura, che riducono il rischio di configurazioni insicure o non conformi a determinati standard e policy.

Un ulteriore vantaggio dell'IaC per la sicurezza è la possibilità di integrare controlli automatici nello sviluppo di codice dell'infrastruttura. Poiché l'infrastruttura è descritta come codice, essa può essere sottoposta ad analisi statica e verifiche automatiche prima del dispiegamento. Strumenti di analisi consentono di rilevare vulnerabilità, misconfigurazioni e violazioni di policy direttamente nei template che definiscono l'infrastruttura prima che essa venga effettivamente creata e dispiegata in ambienti critici, come quello di produzione [32]. Inoltre, l'automazione del provisioning dell'infrastruttura consente l'integrazione di controlli di sicurezza e verifiche di conformità nelle pipeline di distribuzione, come quella CI/CD, permettendo di bloccare configurazioni errate o non conformi alle policy prima del dispiegamento. Questo approccio introduce la sicurezza direttamente all'interno del processo di sviluppo dei template per la definizione delle infrastrutture. A differenza delle configurazioni manuali tradizionali, che diventano osservabili solo dopo il dispiegamento di risorse, l'IaC consente di esaminare le configurazioni direttamente nel codice ancora prima del provisioning, permettendo quindi di individuare potenziali vulnerabilità in modo preventivo [32]. È possibile, pertanto, eseguire l'analisi per identificare configurazioni insicure, permessi eccessivi ad utenti, esposizioni di rete di servizi non necessari o credenziali incorporate all'interno del codice. Ciò permette di correggere errori e vulnerabilità prima che diventino parte dell'infrastruttura, migliorando significativamente la sicurezza soprattutto in modo preventivo.

3.2.3 Policy as Code

L'IaC abilita inoltre l'applicazione automatica e coerente di policy di sicurezza tramite il paradigma "Policy as Code", in cui i requisiti di sicurezza vengono espressi anch'essi come codice e applicati automaticamente alle configurazioni infrastrutturali [33]. In generale, con Policy as Code si intende l'approccio secondo cui le politiche operative, di sicurezza o di conformità vengono espresse in forma di codice e integrate direttamente nei processi automatizzati di gestione dell'infrastruttura e anche delle applicazioni. Le policy vengono definite in linguaggi specifici e utilizzate dai tool di orchestrazione o provisioning dell'infrastruttura, consentendo in tal modo la verifica automatica e preventiva del rispetto degli standard e dei requisiti di sicurezza prima che le configurazioni vengano effettivamente applicate. Il modello di Policy as Code si collega e opera sullo stesso livello dell'IaC. Infatti se l'Infrastructure as Code utilizza codice per definire la composizione di un'infrastruttura, la Policy as Code introduce un insieme di vincoli e controlli definiti come codice che determinano quali configurazioni siano ammesse e quali debbano essere bloccate o segnalate. Ciò indica quindi che i requisiti come l'esposizione di porte di rete non necessarie, le configurazioni di accesso a determinati sistemi, l'utilizzo di determinati protocolli crittografici o l'isolamento delle risorse, possono essere formalizzati definendo delle regole applicate automaticamente durante il provisioning. Ad esempio, una policy può imporre che solo determinate applicazioni possano esporre servizi su reti pubbliche oppure che tutte le istanze debbano avere configurazioni di sicurezza specifiche.

Durante il processo di dispiegamento, i template IaC, che contengono le configurazioni dell'infrastruttura, vengono analizzati automaticamente dal motore dei tool PaC e se le configurazioni definite nel codice rispettano le policy allora il provisioning viene approvato. In caso contrario, il provisioning viene bloccato o segnalato. Questo procedimento è possibile grazie all'integrazione di questi tool di controllo delle policy all'interno del processo di provisioning e di distribuzione. Un aspetto fondamentale è che le policy diventano versionabili, riutilizzabili e possono essere sottoposte a revisione, esattamente come il codice di configurazione per l'IaC. Ciò permette un controllo maggiore e tracciabile delle modifiche alle regole di sicurezza nel tempo, grazie al controllo delle versioni.

3.2.4 Riduzione del Configuration Drift e Adozione di Infrastruttura Immutabile

L'adozione dell'IaC contribuisce inoltre a ridurre il fenomeno del "configuration drift", ovvero la differenza tra la configurazione prevista e la configurazione reale ed attuale applicata ai sistemi, molto spesso causata dalle modifiche manuali non registrate piuttosto

che tramite i tool di provisioning opportuni. La gestione dichiarativa e automatizzata dell'infrastruttura riduce questo fenomeno di drift e mantiene i sistemi coerenti con le configurazioni definite nei template. Dal punto di vista della sicurezza, invece, la riduzione del drift limita la possibilità che alcuni sistemi o risorse dell'infrastruttura differiscano rispetto alle proprie configurazioni definite nel codice, ritenute sicure tramite processi come il PaC, permettendo quindi che le modifiche vengano eseguite esclusivamente tramite gli appropriati tool di provisioning in modo da creare versioni dei template e di tracciare le modifiche di configurazione applicate nel tempo. Infine, l'IaC supporta il paradigma di infrastruttura immutabile, in cui l'infrastruttura non viene modificata direttamente nel suo ambiente di provisioning ma viene ricreata a partire da configurazioni versionate, come illustrato in 2.3.2. Pertanto, questo modello elimina le modifiche dirette e manuali alle risorse e garantisce che ogni istanza dell'infrastruttura derivi da una definizione controllata e versionata, eliminando così il fenomeno di configuration drift rispetto alle risorse. L'infrastruttura immutabile riduce dunque la superficie di attacco, dovuta a configurazioni errate e modifiche dirette e non controllate che portano potenzialmente all'esposizione di vulnerabilità, e impedisce modifiche manuali non tracciate [34].

Nel complesso, viene evidenziato come l'Infrastructure as Code migliori la sicurezza delle infrastrutture attraverso la standardizzazione e il versionamento delle configurazioni, l'automazione dei processi, la tracciabilità delle modifiche, l'integrazione dei controlli di sicurezza durante la fase di sviluppo delle configurazioni e la verificabilità preventiva di quest'ultime tramite l'utilizzo di policy specifiche. Queste proprietà contribuiscono quindi a ridurre errori umani, configurazioni errate e fenomeni di configuration drift, migliorando la postura generale di sicurezza delle infrastrutture definite da software.

3.2.5 Vulnerabilità e Rischi di Sicurezza dell'Infrastructure as Code

Nonostante i numerosi vantaggi introdotti dall'Infrastructure as Code in termini di automazione, riproducibilità e integrazione della sicurezza nei processi di gestione dell'infrastruttura, l'adozione di tale paradigma introduce anche nuove superfici di attacco e vulnerabilità specifiche.

Misconfigurazioni e Vulnerabilità dei Template IaC

Uno dei principali rischi di sicurezza dell'IaC riguarda la presenza di configurazioni insicure e vulnerabilità direttamente nei template infrastrutturali. Poiché l'infrastruttura viene generata automaticamente a partire dal codice, eventuali errori presenti nel codice vengono replicati regolarmente in tutte le istanze dell'infrastruttura ogni volta

che tale codice viene applicato per il provisioning. I template IaC possono contenere problematiche analoghe a quelle del software, inclusi errori di configurazione, violazioni di policy e parametri non sicuri [32]. Tali difetti possono tradursi in vulnerabilità infrastrutturali concrete all'interno delle risorse, come esposizioni di servizi o protocolli di rete non intenzionali o permessi eccessivi nelle risorse cloud. La caratteristica di automazione del provisioning amplifica l'impatto di tali errori e misconfigurazioni, in quanto una configurazione errata o insicura può essere distribuita rapidamente su numerose risorse dell'infrastruttura. La codifica dell'infrastruttura comporta infatti che errori di configurazione e procedimenti insicuri, tipiche dello sviluppo software, vengano trasferiti direttamente nei sistemi e nelle risorse con impatti potenzialmente estesi su tutta l'infrastruttura e ripetibili, siccome, ad esempio, il provisioning di un'infrastruttura definita da un template con configurazioni errate propagherà gli errori ogni volta che tale template viene applicato. Molti studi mostrano che i template IaC presentano frequentemente vulnerabilità e misconfigurazioni, spesso propagate automaticamente su larga scala a causa della caratteristica del riutilizzo e dell'automazione dell'infrastruttura definita come codice. Il "Threat Report" di Palo Alto Networks del 2020 [35] evidenzia come siano stati individuati più di 199.000 template in uso con vulnerabilità medio-alte al livello di codice, mostrando anche che circa il 65% degli incidenti cloud sia dovuto proprio a semplici misconfigurazioni. Configurazioni errate, come storage pubblici, porte aperte o permessi "Identity and Access Management" (IAM) eccessivi, che possono essere introdotte direttamente nei template IaC, sono particolarmente critici anche negli ambienti cloud, dove il provisioning di risorse è spesso controllato esclusivamente tramite configurazioni dichiarative.

Esposizione di Segreti nei Template IaC

Un rischio particolarmente alto riguarda la gestione e l'inserimento delle credenziali e dei segreti all'interno di codice infrastrutturale. Credenziali, chiavi API o password vengono frequentemente inserite direttamente nei template, rendendole esposte nei repository o nei sistemi di controllo delle versioni. La presenza di credenziali "hard-coded", cioè definite direttamente all'interno del codice, rappresenta una delle categorie più comuni di vulnerabilità nei template IaC [32]. Analogamente, la gestione impropria dei segreti costituisce una criticità, poiché i template spesso includono informazioni sensibili necessarie al provisioning delle risorse [31].

Automazione e Distribuzione dell'Infrastruttura

Un ulteriore aspetto critico riguarda l'automazione dell'infrastruttura stessa, che può amplificare il rischio e l'impatto di errori e vulnerabilità. Se il provisioning o le pipeline di

distribuzione vengono compromessi, allora un attaccante può potenzialmente introdurre delle configurazioni malevole all'interno dei template durante tali processi che verranno distribuite automaticamente su larga scala in tutta l'infrastruttura e in tutti gli ambienti in cui essa deve essere dispiegata. Questi processi richiedono quindi determinati controlli di sicurezza, poiché la compromissione dell'automazione del provisioning e della distribuzione può comportare dei rischi sull'intera infrastruttura [34].

Sono inoltre presenti rischi sostanziali ai sistemi di controllo delle versioni. Poiché l'infrastruttura è gestita come codice, i repository, in cui vengono memorizzati tutti i template che costituiscono le versione di una certa infrastruttura, diventano dei componenti critici, siccome la loro compromissione può consentire a un attaccante di modificare direttamente il codice nei template per renderli volutamente vulnerabili in modo da attuare delle modifiche malevoli e persistenti [31].

Rischi per la Supply Chain

Infine, la possibilità di definire parti di un'infrastruttura in moduli riutilizzabili pone un'ulteriore rischio se tali moduli sono vulnerabili, siccome la loro distribuzione e riutilizzo in altri progetti può portare ad una problematica di "Supply Chain". Infatti, L'IaC introduce rischi legati alla fiducia nei moduli e negli strumenti di terze parti utilizzati per il provisioning. I template IaC dipendono spesso da provider e possono importare moduli esterni, che possono contenere vulnerabilità o comportamenti insicuri. Il mondo dell'IaC include numerose dipendenze e componenti esterni che possono costituire fonti di vulnerabilità. Poichè la sicurezza dell'infrastruttura dipende anche dall'integrità e dalla sicurezza di tali componenti, tutto ciò pone appunto un rischio di vulnerabilità relativo alla Supply Chain [32].

Nel complesso, l'Infrastructure as Code introduce nuove categorie di vulnerabilità legate alla codifica dell'infrastruttura, alla gestione dei segreti, alla caratteristica di riutilizzo dei moduli, alla sicurezza dei processi di provisioning e di distribuzione e alla sicurezza della Supply Chain per i componenti che vengono utilizzati. Sebbene il paradigma dell'IaC migliori la sicurezza in termini di automazione e controllo, esso trasferisce nell'infrastruttura i tipici rischi e vulnerabilità di un software, essendo definita tramite codice. Di conseguenza, la sicurezza del modello IaC dipende non solo dalle configurazioni definite nel codice, ma anche dalla sicurezza dello sviluppo dell'infrastruttura e dei processi automatizzati che la gestiscono.

Capitolo 4

Implementazione dei casi di studio

In questo capitolo vengono mostrate le implementazioni, usando il codice Terraform, dei casi di studio che rappresentano le segmentazioni dell'infrastruttura della rete OT, descrivendo anche le scelte implementative prese durante lo sviluppo. Verrà presentata la struttura generica del codice ed infine saranno mostrate le implementazioni specifiche per ogni tipo di segmentazione.

4.1 Presentazione delle Scelte Implementative

Il progetto è suddiviso come segue:

- Vengono create tre diverse directory che contengono le tre implementazioni, presentate successivamente in questo capitolo, ognuna delle quali rappresenta distintamente un progetto Terraform.
- Ogni directory contiene quattro file identici, il "main.tf", "sdn_resources.tf", "vm_resources.tf" e infine il file "variables.tf", creati per definire la struttura del progetto stesso e che verranno descritti in seguito.
- Il file "terraform.tfvars", invece, è unico per ogni directory e contiene le istanze delle variabili che verranno poi mappate sulle rispettive risorse Terraform. Questo file corrisponderà ad ogni specifica implementazione, come viene descritto nella sottosezione 4.1.4.

4.1.1 Gestione e Scelta del Provider

Inizialmente, è presente il file principale "main.tf" in cui vengono definiti i provider utilizzati in tutto il progetto Terraform. Come descritto nel capitolo dei preliminari, un provider è essenzialmente un middleware tra Terraform, e il servizio vero e proprio

che si intende utilizzare. Esso è proprio il componente responsabile delle chiamate API al servizio stesso, che in questo caso di studio sarebbe Proxmox. Il provider fornisce i tipi di risorse da utilizzare nel codice Terraform per interagire con il servizio. La scelta del provider è cruciale e dipende molto dalle necessità del progetto Terraform. Facendo un esempio, un provider per Proxmox è Telmate che si occupa di gestire unicamente il provisioning per le macchine virtuali e i container, non coprendo invece tutta la parte della SDN Proxmox. Di fatti, Telmate fornisce un numero minimale di risorse le cui principali sono "lxc" e "vm_qemu", per gestire rispettivamente i container e le macchine virtuali all'interno di Proxmox, tuttavia non presenta alcuna risorsa per la gestione della rete virtuale. Per questo motivo, il provider Telmate è più adatto a progetti che richiedono esclusivamente la gestione delle macchine piuttosto che la rete. In questo caso di studio, il provider scelto per l'implementazione è invece "BPG", che è molto più completo di Telmate e copre quasi tutte le chiamate API verso Proxmox, incluse quelle per la SDN. Infatti, esso presenta molte più risorse rispetto a Telmate, comprese quelle per la rete virtuale, proprio per questo motivo è il più adatto al progetto del caso di studio presentato.

Successivamente, viene definito l'accesso ai provider tramite l'utilizzo di credenziali oppure token di autenticazione, dipende dai metodi supportati dai provider stessi per eseguire l'accesso, per poter utilizzare le API della piattaforma con cui si comunica. In questo specifico caso viene utilizzato il secondo metodo.

```
terraform {
    required_providers {
        proxmox = {
            source = "bpg/proxmox"
        }
    }
}

provider "proxmox" {
    endpoint      = var.proxmox_url
    api_token     = "${var.token_id}=${var.token_secret}"
    insecure      = true
}
```

Come anticipato, vengono specificati i provider che saranno utilizzati per l'intero progetto Terraform, come in questo caso il provider "BPG" per Proxmox. Successivamente, si definiscono i parametri necessari che serviranno al provider per comunicare tramite le API del servizio che si intende utilizzare, come ad esempio il parametro "endpoint" che specifica il nodo del Cluster Proxmox a cui connettersi e il parametro "api_token" contenente il token generato sulla piattaforma di Proxmox che permette l'autenticazione. Infine, il parametro "insecure" è utilizzato in modo tale da offuscare i parametri sensibili, come appunto il token oppure una password se si utilizza l'accesso tradizionale con username.

4.1.2 Definizione dell'infrastruttura di rete

Il file "sdn_resources.tf", invece, è stato creato per raggruppare tutte le risorse Terraform, definite dal provider BPG, che rappresentano unicamente l'intera SDN di Proxmox In particolare, vengono fornite le risorse per i 5 tipi di zone presenti nella SDN, una risorsa che rappresenta la VNet, ed infine una risorsa per la Subnet. Perciò, utilizzando queste risorse è possibile configurare l'intera SDN di Proxmox, completamente gestibile tramite codice e in modo automatico.

```
resource "proxmox_virtual_environment_sdn_zone_simple" "zona_simple" {
    for_each = var.zone_simple

    id    = each.value.id
    ipam  = each.value.ipam
    nodes = each.value.nodes
    dhcp  = lookup(each.value, "dhcp", null)
    dns   = lookup(each.value, "dns", null)
    mtu   = lookup(each.value, "mtu", null)

    depends_on = [
        proxmox_virtual_environment_sdn_applier.finalizer
    ]
}
```

```
resource "proxmox_virtual_environment_sdn_vnet" "internalNet" {
    for_each = var.vnet

    id          = each.value.id
    zone        = each.value.zone
    vlan_aware  = lookup(each.value, "vlan_aware", null)
    tag         = lookup(each.value, "tag", null)

    depends_on = [
        proxmox_virtual_environment_sdn_applier.finalizer,
        proxmox_virtual_environment_sdn_zone_simple.zona_simple
    ]
}

resource "proxmox_virtual_environment_sdn_subnet" "subnet" {
    for_each = var.subnet

    cidr = each.value.cidr
    vnet = each.value.vnet

    gateway = lookup(each.value, "gateway", null)

    dhcp_range = {
        start_address = lookup(each.value.dhcp_range, "start_address", null)
        end_address = lookup(each.value.dhcp_range, "end_address", null)
    }

    dhcp_dns_server = lookup(each.value, "dhcp_dns_server", null)
    snat = lookup(each.value, "snat", null)

    depends_on = [
        proxmox_virtual_environment_sdn_applier.finalizer,
        proxmox_virtual_environment_sdn_vnet.internalNet,
    ]
}
```

Ogni blocco è costituito da attributi che consentono di istanziare e configurare una risorsa virtuale. Considerando il codice appena presentato, la risorsa che rappresenta la "zona simple" di Proxmox contiene dei campi obbligatori come "id", una stringa che identifica univocamente la zona all'interno della rete virtuale, e il campo "nodes" che specifica a quali nodi, o quale nodo, la zona verrà assegnata. Considerando invece la VNet, anch'essa contiene un campo "id", analogamente alla zona, e un campo "zone" che deve obbligatoriamente contenere l'id di una zona alla quale la VNet verrà assegnata. Analogamente per la risorsa subnet, è presente un campo "cidr", che specifica l'indirizzo

ip della subnet in formato CIDR, e un campo “vnet”, anch’esso rappresentante l’id della VNet a cui questa risorsa subnet deve essere assegnata nella rete Proxmox. Ogni risorsa contiene anche dei parametri opzionali i cui valori sono indicati tramite l’utilizzo della funzione “lookup()”, che controlla se un determinato campo contiene effettivamente un valore oppure no. Se non è presente alcun valore, Terraform utilizzerà il valore di default impostato dal provider, se presente, altrimenti un valore nullo.

Come si può notare dal codice e dalla descrizione appena fornita, tra queste risorse sono presenti delle dipendenze. Più nello specifico la VNet dipende dalla zona, siccome un suo campo è relativo all’id di quest’ultima, mentre la subnet dipende dalla VNet, per lo stesso motivo. Pertanto, viene creata una catena di dipendenze siccome la prima ad essere creata deve essere la zona, successivamente viene creata la VNet da assegnare alla zona, ed infine viene creata la Subnet assegnata a sua volta alla VNet. In questo modo si può dare un ordine di creazione delle risorse a Terraform che, altrimenti, genererebbe tutte le risorse in parallelo. Quindi, nel caso in cui non si definissero delle dipendenze tra queste risorse potrebbe accadere che durante l’applicazione del plan di Terraform, ovvero proprio quando devono essere create tutte le risorse specificate nel codice, alcune risorse vengano create prima di altre e se ciò avviene allora vengono restituiti degli errori poichè una risorsa, che ha come attributo interno un collegamento con un altra risorsa che non è ancora stata creata, non può essere generata poichè non è presente un attributo obbligatorio che permetta la corretta creazione della risorsa stessa.

In altre parole, facendo un esempio, se non fosse specificata alcuna dipendenza, potrebbe accadere che la creazione della VNet avvenga prima della zona. Siccome la VNet richiede come attributo interno l’id della zona, che è obbligatorio, a cui essa deve essere assegnata, allora Terraform restituisce un errore, specificando che un certo attributo obbligatorio è nullo, cioè proprio l’id della zona.

La stessa questione vale tra la VNet e la Subnet. Quindi, se Terraform creasse una risorsa subnet prima di una risorsa VNet, allora l’id della VNet, che deve obbligatoriamente essere presente come attributo all’interno della risorsa subnet, sarebbe nullo, il chè porterà necessariamente ad un errore e, di conseguenza, all’interruzione della creazione dell’infrastruttura Terraform. Perciò, proprio per questi motivi appena illustrati, le dipendenze sono dei fattori cruciali nella creazione di infrastrutture tramite Terraform.

Oltre alle risorse appena mostrate, ne esiste un'ulteriore che simula il funzionamento dell'applicazione delle modifiche alla SDN interna di Proxmox. Tale risorsa è chiamata appunto "applier".

```
    resource "proxmox_virtual_environment_sdn_applier" "apply" {
    lifecycle {
        replace_triggered_by = [
            proxmox_virtual_environment_sdn_zone_simple.zona_simple,
            proxmox_virtual_environment_sdn_vnet.internalNet,
            proxmox_virtual_environment_sdn_subnet.subnet,
        ]
    }

    depends_on = [
        proxmox_virtual_environment_sdn_zone_simple.zona_simple,
        proxmox_virtual_environment_sdn_vnet.internalNet,
        proxmox_virtual_environment_sdn_subnet.subnet,
    ]
}


resource "proxmox_virtual_environment_sdn_applier" "finalizer" {}
```

In questo caso, l'applier è l'ultima risorsa creata del file, poichè ha come dipendenze tutte le altre. Concettualmente, questa risorsa agisce in modo da rilevare se sono state eseguite modifiche nelle altre risorse di rete ed avviarsi per contattare le API di Proxmox, che applica appunto le modifiche alla SDN. Viene utilizzata con "replace_triggered_by" che in pratica permette l'esecuzione della risorsa se sono state eseguite modifiche a ciò che è presente tra le sue dipendenze.

La risorsa applier è una "null resource", ovvero è una risorsa che non viene realmente creata all'interno della piattaforma, ma piuttosto è presente solamente all'interno dello state file di Terraform. Questo è dovuto proprio al funzionamento stesso dell'applier, che deve semplicemente contattare Proxmox per applicare le modifiche alle risorse.

Oltre all'applier, spesso viene creata anche un'altra risorsa, sempre di tipo applier, chiamata "finalizer". Questa risorsa è completamente vuota poichè serve per gestire l'intera gerarchia di dipendenze tra tutte le risorse, e pone se stessa come risorsa centrale come radice dell'albero gerarchico in modo tale da essere creata per prima. E' essenzialmente un modo tecnico per dire a tutte le risorse che devono attendere prima la creazione del finalizer.

4.1.3 Definizione delle macchine virtuali

Infine, è presente il file "vm_resources.tf" contenente la risorsa della Virtual Machine, strutturata come segue:

```
resource "proxmox_virtual_environment_vm" "test" {
for_each = var.VMs

node_name = each.value.node
vm_id     = each.value.id
name      = each.value.name
tags      = each.value.tags

keyboard_layout = lookup(each.value, "keyboard_layout", null)

bios            = lookup(each.value, "bios", null)
machine         = each.value.machine
scsi_hardware   = lookup(each.value, "scsi_hardware", null)

on_boot = false
started = false

operating_system {
    type = each.value.operating_system.type
}

cpu {
    cores   = each.value.cpu.cores
    sockets = each.value.cpu.sockets
    type    = each.value.cpu.type
}

agent {
    enabled = each.value.agent.enabled
}

memory {
    dedicated = each.value.memory.dedicated
    floating  = each.value.memory.floating
}
```

```
    clone {
        vm_id        = each.value.clone.vm_id
        full         = each.value.clone.full
        node_name    = each.value.clone.node_name
        datastore_id = each.value.clone.datastore_id
        retries      = 5
    }

    disk {
        datastore_id    = each.value.disk.datastore_id
        interface       = each.value.disk.interface
        size            = each.value.disk.size
        file_format     = lookup(each.value.disk, "file_format", null)
    }

    network_device {
        bridge      = each.value.network_device.bridge
        mac_address = lookup(each.value.network_device, "mac_address", null)
        model       = lookup(each.value.network_device, "model", null)
        vlan_id     = lookup(each.value.network_device, "vlan_id", null)
    }

    depends_on = [
        proxmox_virtual_environment_sdn_applier.apply,
    ]
}
```

In questo caso specifico, è stata scelta la creazione delle macchine virtuali tramite la clonazione di una macchina template, di distribuzione Linux-Alpine, in modo tale da rendere più veloce il test di dispiegamento delle macchine virtuali.

Nel caso della macchina virtuale, tra le dipendenze è stata inserita la risorsa dell'applier della SDN, in quanto durante la creazione della VM bisogna specificare l'interfaccia di rete da assegnare ad essa. Pertanto, se ad una VM viene assegnata una VNet come scheda di rete virtuale, potrebbe capitare che in Terraform, durante la creazione dell'infrastruttura, la risorsa della VNet in questione venga creata dopo la macchina virtuale stessa e, se ciò accade, Terraform cercherà l'id della VNet all'interno del suo state file senza però riuscire a trovarla, in quanto non è ancora stata creata. Perciò, tale mancanza può portare ad un problema di dipendenza tra le risorse. Quindi per evitare ciò, si aggiunge la risorsa dell'applier come dipendenza alla macchina virtuale per assicurare che la macchina venga creata dopo tutte le risorse che costituiscono la SDN, impedendo così tale problema. E' esattamente la stessa questione delle dipendenze tra le risorse della SDN descritta in precedenza.

4.1.4 Esempi e Spiegazioni

I tre file descritti finora contengono delle strutture comuni a tutte le implementazioni, in modo tale da rendere scalabile il processo di scrittura del programma, evitando così la definizione ogni volta di tante risorse manualmente.

Tradizionalmente in Terraform, la creazione di una risorsa viene eseguita tramite la sua definizione nel codice, in modo da rispecchiare esattamente come la sua controparte virtuale deve essere. Vengono quindi specificati tutti gli attributi obbligatori ed eventualmente omessi quelli opzionali, viene inoltre specificato il tipo di risorsa ed il nome per identificarla univocamente. Infine, vengono definite, se presenti, delle dipendenze relative ad altre risorse, come mostrato precedentemente. Questo procedimento viene fatto manualmente durante la definizione di una singola risorsa e deve essere reiterato nel caso in cui debbano essere definite più risorse.

Tuttavia, in questo caso di studio, la scelta implementativa è diversa. Più nello specifico, viene definita una singola risorsa di un certo tipo che rappresenta la sua struttura generica e che è in grado di adattarsi in base alle necessità, definendo tutti gli attributi obbligatori che devono essere presenti al suo interno, e definendo attributi opzionali che possono essere compilati oppure no. Questa struttura viene utilizzata come contenitore per tutte le risorse dello stesso tipo che devono essere create, ognuna delle quali sarà ovviamente distinta da un attributo id. All'interno di questa risorsa generica è presente un foreach che itera su una variabile, presente all'interno del file "terraform.tfvars", contenente una mappa di oggetti. Ogni oggetto rappresenta, appunto, una specifica risorsa da creare che contiene i valori degli stessi attributi presenti nella risorsa generica. In questo modo, ogni oggetto sarà quindi una istanza specifica per la risorsa generica corrispondente. Ogni insieme è relativo ad una specifica struttura di risorsa definita precedentemente.

Nel caso dell'infrastruttura di rete, si hanno le risorse per la VNet e per la subnet, le cui strutture sono sempre le stesse in ogni caso implementato. Questo suggerisce che è possibile creare due risorse generiche, rappresentanti la VNet e la subnet, che fungeranno da scheletri per la creazione di tutte le istanze specificate nelle rispettive variabili. Quindi, se in un'infrastruttura è richiesta la creazione di un certo numero di VNet e subnet, esse verranno definite come degli oggetti e iterate nel foreach delle rispettive risorse generiche, in modo da creare tutte le istanze automaticamente.

Tuttavia, se sono presenti delle risorse con strutture diverse, non è possibile applicare questo tipo di implementazione. Facendo un esempio, le risorse delle zone Proxmox sono cinque tipi di risorse distinte, siccome ognuna di esse varia per contenuto e tipo di attributi. Pertanto se è richiesta una zona di tipo Simple, bisognerà creare una risorsa generica che rispecchi quel tipo di zona specifica, mentre se è richiesta una zona di tipo

VXlan allora sarà inevitabilmente necessaria la creazione di un'altra risorsa generica rappresentante la zona VXlan.

Questa modalità di definizione del codice di un'infrastruttura permette di creare più comodamente molteplici risorse come dei semplici oggetti contenenti determinati valori e definire, invece, la risorsa effettiva una sola volta.

Per fare un esempio, la creazione delle zone viene definita tramite una variabile come segue:

```
variable "zone_simple" {
    type = map(object({
        id = string
        ipam = string
        nodes = list(string)
        dhcp = optional(string)
        dns = optional(string)
        mtu = optional(number)
    }))
}
```

La variabile "zone_simple", presente nel file "variables.tf", contiene quindi una mappa di oggetti che definisce la struttura di ogni risorsa. Nell'esempio mostrato sopra, la variabile che definisce la zona di tipo simple contiene tutti gli attributi che devono essere presenti anche all'interno della risorsa stessa. In questo modo, si può popolare la variabile per poter creare un'insieme di oggetti, ognuno dei quali rappresenta un'istanza specifica della risorsa generica presente nel file "sdn_resources.tf". Lo stesso vale per la risorsa delle macchine virtuali contenuta nel file "vm_resources.tf".

L'assegnamento dei valori di ogni variabile avviene nel file "terraform.tfvars", file globale del modulo root, che deve essere unico per tutta la gerarchia dei moduli. All'interno di questo file sono definite le istanze per ogni variabile.

L'esempio della zona è il seguente:

```
zone_simple = {
    "zone1" = { id = "IDMZ", ipam = "pve", nodes = ["pve"],
                mtu = 1500, dhcp = "dnsmasq" }

    "zone2" = { id = "OPS", ipam = "pve", nodes = ["pve"],
                mtu = 1500, dhcp = "dnsmasq" }

    "zone3" = { id = "PROD", ipam = "pve", nodes = ["pve"],
                mtu = 1500, dhcp = "dnsmasq" }
}
```

Come viene mostrato, "zone_simple" è la variabile descritta precedentemente che contiene una mappa di oggetti, in cui ognuno di essi contiene i valori degli attributi presenti nella risorsa corrispondente. Tutti questi oggetti vengono poi assegnati al foreach all'interno della risorsa generica, in modo tale da poter mappare ogni valore dell'oggetto su ogni attributo presente nella risorsa.

Secondo questo esempio, verranno quindi create tre zone identificate da id univoci, tutte e tre presenti sullo stesso nodo. Grazie alla funzione "lookup" descritta in precedenza, è possibile omettere i valori degli attributi opzionali, come ad esempio il campo "mtu" oppure il campo "dhcp". Se essi non sono presenti all'interno dell'oggetto, verranno istanziati con i valori di default forniti dal provider.

4.2 Analisi dei Casi di Studio e Implementazione

In questa sezione vengono mostrate le riproduzioni delle implementazioni relative al caso di studio dell'articolo di Callegati et al. [36]. Concettualmente, il caso di studio si concentra nel creare tre tipi diversi di segmentazione di un'infrastruttura rete "Operational Technology" (OT) di un impianto industriale, per mostrare come esse differiscono rispetto alla resistenza ad attacchi esterni al livello di rete, simulati tramite il tool Caldera MITRE, mostrando come una segmentazione efficiente può migliorare significativamente la tolleranza e resilienza della rete stessa.

Essenzialmente, l'implementazione si basa sulla creazione del Digital Twin dell'infrastruttura OT.

Un Digital Twin, Gemello Digitale, fondamentalmente è una replica fedele virtuale di un sistema fisico, di un processo o persino di un intero sistema complesso con più componenti, come una rete reale [37]. Un Digital Twin può essere aggiornato dinamicamente tramite dati provenienti dal sistema fisico che vengono inviati attivamente alla copia virtuale. Il gemello digitale, dunque, può essere utilizzato specialmente per eseguire test e simulazioni, a volte considerati rischiosi se eseguiti direttamente sul sistema fisico, in modo da poter analizzare eventuali problemi e visualizzare determinati comportamenti migliorando la sua copia originale.

Il caso di studio dell'articolo sfrutta l'utilizzo del gemello digitale per condurre simulazioni di sicurezza, riprodurre scenari reali e applicare modifiche all'architettura della rete, il tutto per permettere di valutare la resilienza della rete reale rispetto ad attacchi informatici.

Le riproduzioni si concentrano principalmente sulla struttura delle segmentazioni di rete, mostrando come è possibile l'automazione del dispiegamento e della gestione di una

rete per la creazione di un Digital Twin, utilizzando il paradigma dell'IaC attraverso il software Terraform.

I tipi di segmentazione vengono riprodotti tramite l'utilizzo di zone, presenti nella SDN di Proxmox, ovvero dei perimetri definiti fisicamente o logicamente all'interno dei quali vengono raggruppati determinati dispositivi. Ogni zona ha un proprio firewall perimetrale attraverso il quale deve obbligatoriamente passare tutto il traffico di rete entrante ed uscente dalla zona stessa. Ogni dispositivo può conoscere solamente altri dispositivi appartenenti alla sua stessa zona. L'intercomunicazione tra zone avviene tramite il firewall perimetrale della zona da cui parte la comunicazione di rete, traffico uscente (egress), e il firewall perimetrale della zona a cui la comunicazione è destinata, traffico entrante (ingress).

4.2.1 Struttura Generale

In particolare, viene utilizzata la zona di tipo Simple, che è essenzialmente un semplice Routing Bridge di livello 3 con Network Address Translation, per poter comunicare con la rete esterna al singolo nodo Proxmox.

In pratica, la creazione della zona della SDN viene eseguita come segue:

1. Per prima cosa, si crea la zona stessa, di tipo Simple, nella quale si specifica quali sono i nodi assegnati ad essa, di conseguenza anche le macchine virtuali presenti su di essi, e quale nodo deve agire da IPAM, cioè IP Address Management, ovvero il nodo che è responsabile della gestione degli indirizzi IP dei client. Un IPAM può essere associato ad una o più zone. Infine si può specificare se una zona può usare o meno il DHCP per l'assegnazione automatica degli indirizzi IP ai client appartenenti a suddetta zona.

2. Successivamente vengono create le VNet, ovvero delle reti virtuali che possono agire da Vlan oppure ospitare una o più subnet, e che vengono poi associate ad una specifica zona. Le VNet vengono assegnate alle Virtual Machines come schede di rete virtuali. In tal modo, si può assegnare una VM ad una Subnet oppure ad una Vlan, in base al tipo della VNet.

3. Infine, si creano una o più Subnet da assegnare alle VNet. In una Subnet viene specificato il range di indirizzi IP, se si utilizza il DHCP oppure no, l'indirizzo del Gateway e l'indirizzo della Subnet stessa

Ogni implementazione conterrà delle macchine virtuali, create per rappresentare i sistemi reali e i processi industriali dell'infrastruttura di rete OT.

Di seguito, vengono elencati i vari tipi di sistemi presenti all'interno della rete OT, descrivendo brevemente le loro funzionalità:

PLC Server : Utilizza il software "OpenPLC", ottiene i dati dei processi industriali e li invia da remoto attraverso la rete.

Operator Laptop : Simula il laptop di un ingegnere o operatore della rete OT che può connettersi ai server PLC.

Human Machine Interface/Local SCADA : Contiene il software open source ScadaBR che viene configurato per leggere i dati provenienti dai PLC.

Remote Access Jump Host : E' configurata per potersi connettere da remoto agli HMI, tramite il software Anydesk.

Data Historians : Viene utilizzata per accedere ai dati dai server Local SCADA.

Inventory Management System : Utilizza una demo di un software open source IMS.

Manufactoring Execution System : Utilizza una demo di un software open source MES.

Engineering Workstation : E' configurata con il software open source Open PLC permettendo agli operatori di configurare i PLC.

L'utilizzo di queste macchine virtuali è puramente a scopo di test per le varie segmentazioni di rete, pertanto, per ogni tipo di macchina, viene utilizzata la distribuzione Linux Alpine con le stesse configurazioni interne del sistema, eccetto ovviamente per i campi identificativi della macchina e la sua scheda di rete virtuale.

Per ogni implementazione, viene fornito il codice del corrispettivo file "terraform.tfvars", ovvero il file contenente tutte le istanze delle risorse che vengono create, seguendo la scelta implementativa descritta in precedenza, e che quindi rappresenta l'intera infrastruttura di ogni riproduzione.

4.2.2 Implementazione 1: Segmentazione Piatta

La prima implementazione della rete OT si basa sull'infrastruttura più comune, che in questo caso prevede una segmentazione di rete basilare. Concretamente, viene implementata con un'unica zona della SDN di Proxmox, protetta dal firewall di perimetro, ospitante tutti i sistemi OT. Pertanto, questa implementazione della rete OT è priva di ogni tipo di ulteriore segmentazione interna, ponendo sistemi con funzionalità diverse

nella stessa zona e sottorete. Tutti i sistemi, rappresentati da macchine virtuali, vengono posizionati all'interno di un singolo nodo Proxmox, appartenente esso stesso alla singola zona di rete creata.

Di seguito vengono mostrate le variabili che compongono la struttura di rete.

```
zone_simple = {
    "zone" = {  id = "FlatOT", ipam = "pve", nodes = ["pve22"],
                mtu = 1500, dhcp = "dnsmasq" }
}

vnet = {
    "vnet" = { id = "single", zone = "FlatOT" }
}

subnet = {
    "subnet" = { cidr = "192.168.1.0/24", vnet = "single",
                 gateway = "192.168.1.254",
                 dhcp_range = {start_address = "192.168.1.2",
                 end_address = "192.168.1.253"},
                 dhcp_dns_server = "192.168.1.254", snat = true }
}
```

L'immagine sottostante descrive esattamente la forma dell'infrastruttura di rete di questa implementazione:

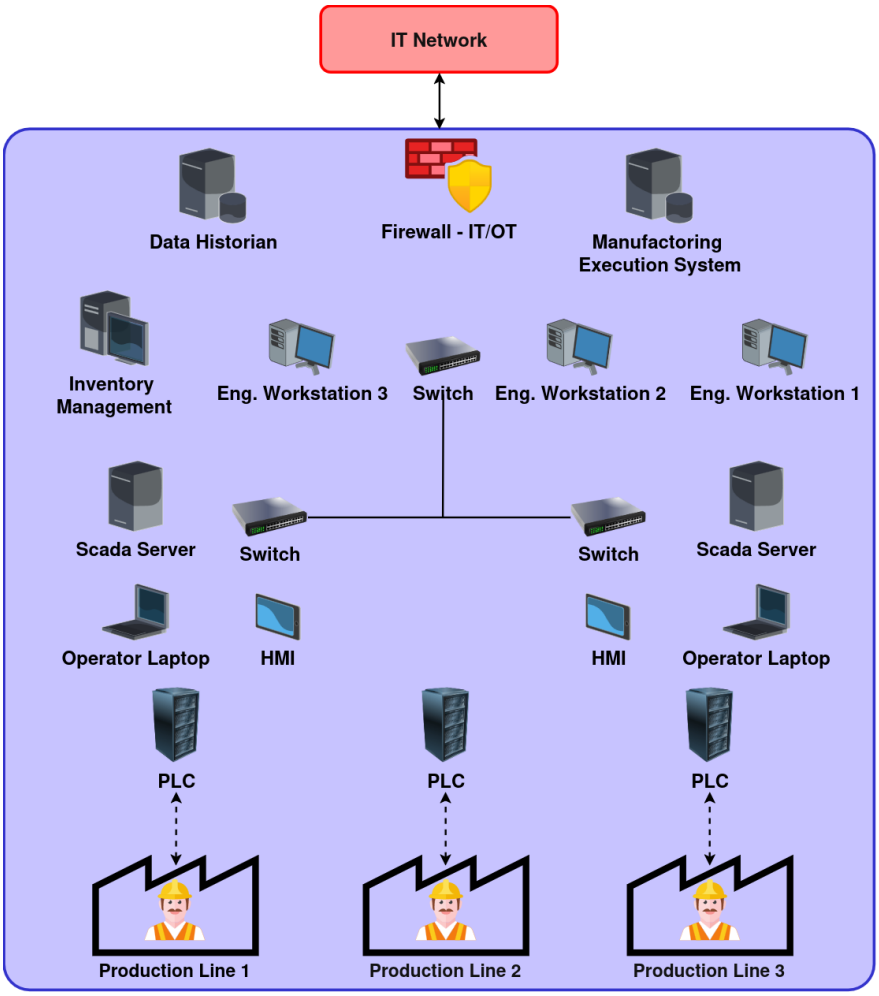

Figura 4.1: Topologia della rete con segmentazione "flat"

Come specificato prima, esiste un'unica zona presente su un solo nodo (pve22), con DHCP abilitato per le sottoreti che verranno assegnate ad essa. Viene creata una sola VNet assegnata alla zona. Ed infine, si crea una subnet, associata a sua volta alla VNet, a cui viene assegnato un range di indirizzi IP, gestibili dal DHCP, e viene definito il gateway della Subnet, che è in grado di contattare la rete al di fuori del nodo corrente grazie all'utilizzo del NAT sorgente "(snat = true)"

Di seguito, il codice che elenca le variabili costituenti le macchine virtuali, che rappresentano i sistemi della rete OT.

```
VMs = {
    "vm1" = { name = "PLC", id = 110, node = "pve22",
              tags = ["alpine", "debian"], keyboard_layout = "it",
              description = "PLC server", machine = "q35",
              operating_system = {type = "l26"},
              cpu = {cores = 1, sockets = 1, type = "host"},
              agent = {enabled = true},
              memory = {dedicated = 512, floating = 512},
              clone = {vm_id = 101, full = true, node_name = "pve22",
              datastore_id = "local-lvm"},
              disk = {datastore_id = "local-lvm", interface = "scsi0",
              size = 8}, network_device = {bridge = "single"}
            },

    "vm2" = {  name = "Laptop", id = 120, node = "pve22", .....
               description = "Operator Laptop",
               ..... network_device = {bridge = "single"}
            },

    "vm3" = {  name = "Scada", id = 130, node = "pve22", .....
               description = "Scada Server",
               ..... network_device = {bridge = "single"}
            },

    "vm4" = {  name = "HMI", id = 140, node = "pve22", .....
               description = "HMI Tablet",
               ..... network_device = {bridge = "single"}
            },

    "vm5" = {  name = "History", id = 150, node = "pve22", .....
               description = "Data Historian DataBase",
               ..... network_device = {bridge = "single"}
            },

    "vm6" = {  name = "Workst", id = 160, node = "pve22", .....
               description = "Eng. Workstation",
               ..... network_device = {bridge = "single"}
            },
}
```

Vengono dunque create sei Virtual Machines che rappresentano in generale tutti i tipi di sistemi presenti all'interno della rete OT. Ogni macchina ha il suo identificativo numerico, un nome e una descrizione che indica il tipo di macchina reale rappresentata e il nodo su cui essa deve essere creata.

Successivamente, il campo “network_device” rappresenta la scheda di rete virtuale della macchina, a cui viene assegnato un bridge collegato alla scheda di rete fisica del nodo Proxmox oppure una VNet. In questo caso specifico è presente una singola rete virtuale assegnata a tutte le macchine.

Ogni implementazione specifica è di questa stessa forma, quindi verrà presentato il codice che costituisce le variabili delle risorse di rete e le variabili che rappresentano le macchine virtuali, esattamente come è stato illustrato per questa implementazione.

In ogni macchina, è presente inoltre il campo “clone”, che rappresenta la serie di informazioni necessarie per poter clonare la macchina template. Ogni macchina virtuale ha gli stessi campi, essendo il template unico. In particolare, vengono specificati:

- L’id della macchina template, da cui eseguire il clone.
- Il campo “full”, che indica se deve essere eseguita una copia completa del disco del template e creare un disco virtuale separato per ogni VM.
- Il “node_name”, che rappresenta il nodo Proxmox su cui è presente la macchina template.
- Infine, il “datastore_id”, ovvero lo storage del nodo Proxmox in cui è memorizzato il disco virtuale della macchina da clonare.

Questa prima implementazione appena descritta, che presenta una “Flat Architecture”, può essere resa più sicura aumentando la segmentazione della rete. La prossima variante utilizza una “segmentazione orizzontale”, ovvero la segregazione dei sistemi OT basata su una gerarchia rispetto alle funzionalità dei sistemi stessi, e anche rispetto ai livelli di località dei processi gestiti.

4.2.3 Implementazione 2: Segmentazione Intermedia

Questa implementazione prevede una segmentazione intermedia orizzontale della rete OT con l'utilizzo di tre zone:

Zona 1: Industrial Demilitarized Zone (IDMZ)

La IDMZ è un componente cruciale per la sicurezza dell'intera rete, che serve come zona cuscinetto tra la rete Information Technology e la rete Operational Technology. Lo scopo principale di questa zona è quello di monitorare e permettere esclusivamente il flusso del traffico di rete consentito, in base a determinate policy, passante tra queste due diverse reti, in modo tale da poter mitigare rischi e incidenti di sicurezza informatica. Essenzialmente, è in grado di proteggere i sistemi critici della rete OT da minacce esterne, permettendo allo stesso modo solamente la normale comunicazione legittima tra le due reti.

I sistemi dunque presenti in questa zona sono i seguenti:

- Una replica del "Data Historian" Database, utilizzato dalla rete IT per recuperare dati importanti processati da sistemi della rete OT
- Un'istanza dedicata del "Active Directory Server" della rete OT, utilizzato per la gestione e definizione di permessi e ruoli all'interno della rete
- Un'istanza MES, utilizzata per monitoraggio e controllo real-time.
- Un'istanza IMS, per il tracciamento e il controllo delle risorse all'interno dell'ambiente di produzione

Zona 2: Operations

La zona operazioni riguarda la supervisione ad alto livello e la gestione delle attività di produzione. In questa zona è presente il server originale "Data Historian", che raccoglie i dati di produzione direttamente dai sistemi locali "SCADA" situati nella zona Produzione. Inoltre, sono presenti delle postazioni che gestiscono i software PLC e che permettono il diretto controllo dei processi di produzione.

Zona 3: Production Zone

Infine, è presente un'unica zona che contiene tutti i sistemi relativi ai processi di produzione:

- I sistemi PLC che gestiscono i dati provenienti e diretti verso i sensori e gli attuatori presenti nella catena di produzione
- Un sistema HMI (Human-Machine Interface), che rappresenta un'interfaccia interattiva tramite la quale l'operatore ha completa visibilità e controllo sulle informazioni di produzione.
- Un client locale per i sistemi SCADA, che acquisisce i dati di produzione dai sistemi PLC e li invia al server SCADA, che sarebbe il server Data Historian, in modo da permettere analisi e monitoraggio
- Un laptop, utilizzato dagli operatori per interagire con i sistemi PLC

Queste zone sono naturalmente implementate tramite l'utilizzo di zone Proxmox, ognuna delle quali ha una subnet dedicata. Ogni zona viene creata su un nodo Proxmox separato.

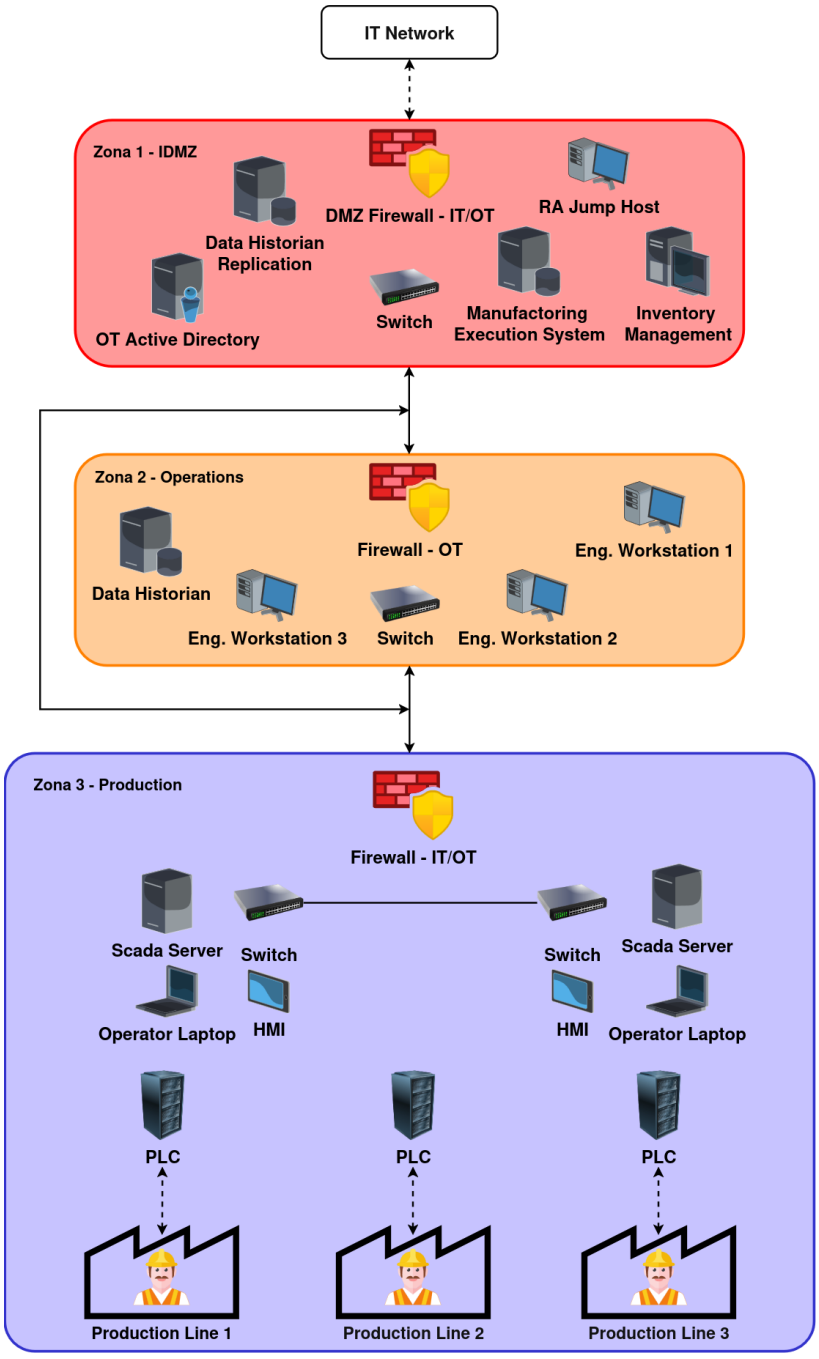

Figura 4.2: Topologia della rete con segmentazione “intermedia”

Di seguito, è presentato il codice Terraform per la riproduzione dell’infrastruttura di rete appena descritta:

```
zone_simple = {
    "zone1" = { id = "IDMZ", ipam = "pve", nodes = ["pve21"],
                mtu = 1500, dhcp = "dnsmasq" }

    "zone2" = { id = "OPS", ipam = "pve", nodes = ["pve22"],
                mtu = 1500, dhcp = "dnsmasq" }
```

```
    "zone3" = { id = "PROD", ipam = "pve", nodes = ["pve23"],
                mtu = 1500, dhcp = "dnsmasq" }
}

vnet = {
    "vnet1" = { id = "idmzNet", zone = "IDMZ" }
    "vnet2" = { id = "opsNet", zone = "OPS" }
    "vnet3" = { id = "prodNet", zone = "PROD" }
}

subnet = {
    "subnet1" = { cidr = "192.168.10.1/24", vnet = "idmzNet",
                  gateway = "192.168.10.254",
                  dhcp_range = {start_address = "192.168.10.10",
                  end_address = "192.168.10.20"},
                  dhcp_dns_server = "192.168.10.254" }

    "subnet2" = { cidr = "192.168.20.1/24", vnet = "opsNet",
                  gateway = "192.168.20.254",
                  dhcp_range = {start_address = "192.168.20.10",
                  end_address = "192.168.20.20"},
                  dhcp_dns_server = "192.168.20.254" }

    "subnet3" = { cidr = "192.168.30.1/24", vnet = "prodNet",
                  gateway = "192.168.30.254",
                  dhcp_range = {start_address = "192.168.30.10",
                  end_address = "192.168.30.50"},
                  dhcp_dns_server = "192.168.30.254" }
}
```

Come mostrato, vengono create tre risorse che rappresentano le zone di tipo simple, ognuna esistente su un nodo Proxmox diverso, abilitando il DHCP in ognuna di esse. Successivamente, vengono create tre VNet, ognuna associata alla zona corrispondente. Infine, si procede alla creazione delle subnet specifiche per ogni zona, in cui vengono definiti i range di indirizzi IP utilizzati dal DHCP.

Infine, è presente il codice per la definizione delle macchine virtuali:

```
VMs = {
    # IDMZ Zone IT/OT
    "vm1" = {   name = "HistRep", id = 110, node = "pve22", .....
                description = "Data Historian (Replication) DB",
                ..... network_device = {bridge = "idmzNet"}
            }

    "vm2" = {   name = "ExecSys", id = 120, node = "pve22", .....
                description = "Manufacturing Execution System DB",
                ..... network_device = {bridge = "idmzNet"}
            }

    "vm3" = {   name = "OTAD", id = 130, node = "pve22", .....
                description = "OT Active Directory",
                ..... network_device = {bridge = "idmzNet"}
            }

    "vm4" = {   name = "RAHost", id = 140, node = "pve22", .....
                description = "RA Jump Host",
                ..... network_device = {bridge = "idmzNet"}
            }

    # Operations Zone OT
    "vm5" = {   name = "History", id = 210, node = "pve23", .....
                description = "Data Historian DB",
                ..... network_device = {bridge = "opsNet"}
            },

    "vm6" = {   name = "Workst", id = 220, node = "pve23", .....
                description = "Eng. Workstation",
                ..... network_device = {bridge = "opsNet"}
            },

    # Production Zone
    "vm7" = {   name = "Scada", id = 310, node = "pve24", .....
                description = "Scada Server",
                ..... network_device = {bridge = "prodNet"}
            },
```

```
    "vm8" = {   name = "Laptop", id = 320, node = "pve24", .....
                description = "Operator Laptop",
                ..... network_device = {bridge = "prodNet"}
            },

    "vm9" = {   name = "HMI", id = 330, node = "pve24", .....
                description = "HMI Tablet",
                ..... network_device = {bridge = "prodNet"}
            },
}
```

La suddivisione delle macchine virtuali all'interno delle zone di rete è relativa alla descrizione del caso di studio, che separa le macchine a seconda del tipo di operazioni che esse eseguono. Precisamente, il database "Data Historian replication", il database "Manufactoring Execution System", il server "Active Directory" e il computer "RA Jump Host" sono tutti inseriti nella prima zona "IDMZ", pertanto ad esse sarà assegnata la VNet "idmzNet" come scheda di rete virtuale e saranno dispiegate tutte sul nodo (22) del cluster Proxmox. In seguito, la macchina "Engineering Workstation" e il database "Data Historian" sono assegnati alla seconda zona "Operations", quindi avranno come scheda di rete virtuale la VNet "opsNet" e saranno poste su un diverso nodo (23). In conclusione, l'ultima zona "Production" sarà popolata dalle macchine "Scada Server", "HMI Tablet" e "Operator Laptop", tutte aventi la VNet "prodNet" come scheda di rete virtuale e posizionate su un terzo nodo (24).

Come è possibile notare, viene eseguita una segmentazione sia a livello di zone di rete, sia come segmentazione fisica dei nodi Proxmox nel cluster, poichè, come anticipato in precedenza, ogni singolo nodo ospita una determinata zona di rete.

4.2.4 Implementazione 3: Micro-Segmentazione

La terza e ultima implementazione riprende la stessa infrastruttura di rete dell'implementazione precedente, aggiungendo però un'ulteriore segmentazione verticale della zona "Operations", dividendola in tre zone differenti e ottenendo così una micro-segmentazione della rete OT [36]. In particolare, la micro-segmentazione è generalmente la proposta più evoluta ed efficace, dimostrando la sua efficienza nel limitare i lateral movement durante o dopo una violazione della rete. Essa è inoltre considerata come fattore chiave per raggiungere un'architettura "Zero Trust", che è progressivamente utilizzata all'interno dei settori industriali.

Come anticipato in precedenza, tutti i sistemi possono comunicare e avere visibilità senza restrizioni esclusivamente su altri sistemi della stessa zona. In più, tutto il traffico

di rete che deve dirigersi verso altre zone, deve passare attraverso il firewall di perimetro della zona di partenza, che serve come porta d'ingresso per il "conduit" che connette due zone, per poi arrivare al firewall perimetrale della zona di destinazione per poter uscire dal conduit e raggiungere i sistemi desiderati. Come anticipato prima, questo è il modello di comunicazione inter-zona utilizzato per mantenere una corretta segmentazione della rete.

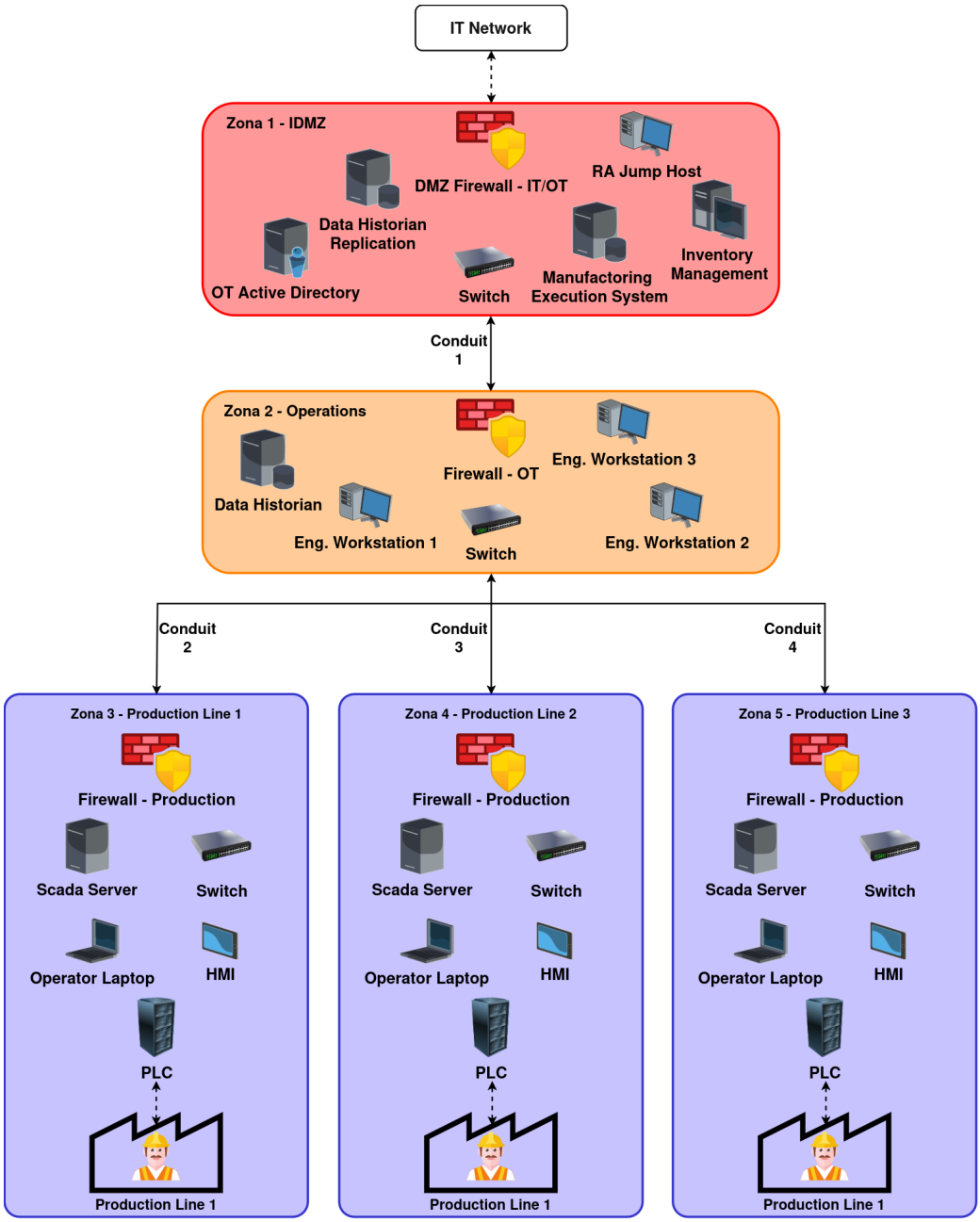

Figura 4.3: Topologia della rete con "micro-segmentazione"

```
zone_simple = {
    "zone1" = { id = "IDMZ", ipam = "pve", nodes = ["pve21"],
                mtu = 1500, dhcp = "dnsmasq" }
    "zone2" = { id = "OPS", ipam = "pve", nodes = ["pve22"],
                mtu = 1500, dhcp = "dnsmasq" }
    "zone3" = { id = "PROD1", ipam = "pve", nodes = ["pve23"],
                mtu = 1500, dhcp = "dnsmasq" }
    "zone4" = { id = "PROD2", ipam = "pve", nodes = ["pve23"],
                mtu = 1500, dhcp = "dnsmasq" }
    "zone5" = { id = "PROD3", ipam = "pve", nodes = ["pve24"],
                mtu = 1500, dhcp = "dnsmasq" }
}

vnet = {
    "vnet1" = { id = "idmzNet", zone = "IDMZ" }
    "vnet2" = { id = "opsNet", zone = "OPS" }
    "vnet3" = { id = "PNet11", zone = "PROD1" }
    "vnet4" = { id = "PNet12", zone = "PROD1" }
    "vnet5" = { id = "PNet21", zone = "PROD2" }
    "vnet6" = { id = "PNet22", zone = "PROD2" }
    "vnet7" = { id = "PNet31", zone = "PROD3" }
    "vnet8" = { id = "PNet32", zone = "PROD3" }
}

subnet = {
    "subnet1" = { cidr = "192.168.10.1/24", vnet = "idmzNet",
                  gateway = "192.168.10.254",
                  dhcp_range = {start_address = "192.168.10.10",
                  end_address = "192.168.10.20"},
                  dhcp_dns_server = "192.168.10.254" }

    "subnet2" = { cidr = "192.168.20.1/24", vnet = "opsNet",
                  gateway = "192.168.20.254",
                  dhcp_range = {start_address = "192.168.20.10",
                  end_address = "192.168.20.20"},
                  dhcp_dns_server = "192.168.20.254" }

    "subnet3" = { cidr = "192.168.30.1/24", vnet = "PNet11",
                  gateway = "192.168.30.254",
                  dhcp_range = {start_address = "192.168.30.10",
                  end_address = "192.168.30.20"},
                  dhcp_dns_server = "192.168.30.254" }

    "subnet4" = { cidr = "192.168.35.1/24", vnet = "PNet12",
                  gateway = "192.168.35.254",
```

```
                dhcp_range = {start_address = "192.168.35.10",
                end_address = "192.168.35.20"},
                dhcp_dns_server = "192.168.35.254" }

    "subnet5" = { cidr = "192.168.40.1/24", vnet = "PNet21",
                gateway = "192.168.40.254",
                dhcp_range = {start_address = "192.168.40.10",
                end_address = "192.168.40.20"},
                dhcp_dns_server = "192.168.40.254" }

    "subnet6" = { cidr = "192.168.45.1/24", vnet = "PNet22",
                gateway = "192.168.45.254",
                dhcp_range = {start_address = "192.168.45.10",
                end_address = "192.168.45.20"},
                dhcp_dns_server = "192.168.45.254" }

    "subnet7" = { cidr = "192.168.50.1/24", vnet = "PNet31",
                gateway = "192.168.50.254",
                dhcp_range = {start_address = "192.168.50.10",
                end_address = "192.168.50.20"},
                dhcp_dns_server = "192.168.50.254" }

    "subnet8" = { cidr = "192.168.55.1/24", vnet = "PNet32",
                gateway = "192.168.55.254",
                dhcp_range = {start_address = "192.168.55.10",
                end_address = "192.168.55.20"},
                dhcp_dns_server = "192.168.55.254" }
}
```

Come è possibile notare dal codice, la segmentazione di rete è aumentata, definendo tre nuove zone in modo da creare una micro-segmentazione che divide la zona Productions dell'implementazione precedente. Così facendo, si ottengono tre nuove mini-zone di produzione, ognuna separata e isolata dalle altre. Di fatto, le macchine virtuali all'interno di una delle tre zone Productions non possono avere visibilità sui sistemi, cioè le altre macchine virtuali, delle altre due zone, rendendo così le tre zone dei processi industriali totalmente separate. Tuttavia, ogni macchina virtuale è in grado di comunicare senza alcuna restrizione con le altre macchine all'interno della stessa zona, quindi il traffico di rete non deve passare attraverso i firewall perimetrali.

Allo stesso modo, le zone IDMZ e Operations, ovvero le prime due zone della rete OT, sono anch'esse segmentate tra loro, ma soprattutto anche da tutte e tre le zone Productions. Tuttavia, come menzionato prima, alcune zone possono avere la necessità di comunicare con altri sistemi in altre zone della rete, pertanto il traffico inter-zona avviene solo ed esclusivamente attraverso i conduits e può avvenire solamente tra zone adiacenti.

Le zone Productions possono comunicare con la zona Operations, e la zona Operations può comunicare anche con la zona IDMZ. Essendo la zona Operations posta nel mezzo dell'infrastruttura di rete, essa può fungere da inoltro per il traffico dalle zone Productions alla zona IDMZ, e viceversa, sempre attraverso l'utilizzo dei conduits. In queste due specifiche implementazioni, le porte d'ingresso dei conduits sono essenzialmente gli stessi nodi Proxmox, dal momento che ogni nodo rappresenta una zona di rete, e lo stesso vale per le porte d'uscita. In questo modo, è possibile configurare le tabelle di routing di ogni nodo per gestire l'inoltro del traffico di rete tra zone adiacenti. Illustrando con un esempio, il nodo 21, che rappresenta la zona IDMZ, può comunicare e inoltrare il traffico al nodo 22, ovvero la zona Operations. Allo stesso modo, le zone Productions, rappresentate dai nodi 23 e 24, possono anch'esse inoltrare il traffico al nodo 22. Infine, la zona Operations, come anticipato prima, è l'unica a poter eseguire il forwarding del traffico verso tutte le altre zone, grazie alla sua posizione centrale all'interno della rete.

Oltre alle regole di routing, vengono definite anche delle regole per i firewall di ogni nodo utilizzando il software IPTables, essendo comunque Proxmox una distribuzione Linux. In questo modo si implementa correttamente il concetto di conduit e di firewall perimetrale, permettendo il traffico di rete solamente tra le macchine che necessitano di una comunicazione inter-zona. La Sezione 4.3 descrive interamente il funzionamento dei firewall all'interno di Proxmox, come possono essere configurati e presenta inoltre le varie implementazioni delle regole per i tipi di segmentazione intermedia e quella descritta nella sezione attuale.

In questa implementazione per la creazione della micro-segmentazione, tramite le tre nuove zone Productions, è stato necessario utilizzare due VNet per ogni zona. Il motivo è molto semplice ed è legato alla gestione delle subnet da parte di Proxmox. Infatti, in Proxmox è possibile assegnare molteplici subnet ad una VNet. Tuttavia, se viene utilizzato il DHCP per l'assegnazione automatica degli indirizzi ip, le macchine virtuali che saranno poste all'interno di una VNet con più subnet, saranno tutte assegnate automaticamente alla prima subnet definita, e mai alle altre. In poche parole, Proxmox non permette di scegliere la subnet a cui assegnare una macchina virtuale tramite DHCP. Pertanto, è stato necessario definire due VNet per ogni zona Productions, e poi assegnare una sola subnet a ogni VNet, in modo tale da ricreare la segmentazione correttamente. Questo è un vincolo di Proxmox che al momento non può essere aggirato.

Per concludere, di seguito è presente il codice per la definizione delle macchine virtuali:

```
VMs = {
# IDMZ Zone IT/OT
"vm1" = {   name = "HistRep", id = 110, node = "pve21", .....
            description = "Data Historian (Replication) DB",
            ..... network_device = {bridge = "idmzNet"} },

"vm2" = {   name = "RAHost", id = 120, node = "pve21", .....
            description = "Remote Access Jump Host",
            ..... network_device = {bridge = "idmzNet"} },

# Operations Zone OT
"vm3" = {   name = "History", id = 210, node = "pve22", .....
            description = "Data Historian DB",
            ..... network_device = {bridge = "opsNet"} },

"vm4" = {   name = "Workst", id = 220, node = "pve22", .....
            description = "Eng. Workstation",
            ..... network_device = {bridge = "opsNet"} },

# Production Zone 1
    # Subnet 1
"vm5" = {   name = "Scada", id = 310, node = "pve23", .....
            description = "Scada Server",
            ..... network_device = {bridge = "PNet11"} },

"vm6" = {   name = "Laptop", id = 320, node = "pve23", .....
            description = "Operator Laptop",
            ..... network_device = {bridge = "PNet11"} },

"vm7" = {   name = "HMI", id = 330, node = "pve23", .....
            description = "HMI Tablet",
            ..... network_device = {bridge = "PNet11"} },

    # Subnet 2
"vm8" = {   name = "Assembly", id = 340, node = "pve23", .....
            description = "Assembly Line 1",
            ..... network_device = {bridge = "PNet12"} },
```

```
    # Production Zone 2
        # Subnet 1
    "vm9" = {   name = "Scada", id = 410, node = "pve23", .....
                description = "Scada Server",
                ..... network_device = {bridge = "PNet21"} },

    "vm10" = {  name = "Laptop", id = 420, node = "pve23", .....
                description = "Operator Laptop",
                ..... network_device = {bridge = "PNet21"} },

    "vm11" = {  name = "HMI", id = 430, node = "pve23", .....
                description = "HMI Tablet",
                ..... network_device = {bridge = "PNet21"} },

        # Subnet 2
    "vm12" = {  name = "Assembly", id = 440, node = "pve23", .....
                description = "Assembly Line 2",
                ..... network_device = {bridge = "PNet22"} },

    # Production Zone 3
        # Subnet 1
    "vm13" = {  name = "Scada", id = 510, node = "pve24", .....
                description = "Scada Server",
                ..... network_device = {bridge = "PNet31"} },

    "vm14" = {  name = "Laptop", id = 520, node = "pve24", .....
                description = "Operator Laptop",
                ..... network_device = {bridge = "PNet31"} },

    "vm15" = {  name = "HMI", id = 530, node = "pve24", .....
                description = "HMI Tablet",
                ..... network_device = {bridge = "PNet31"} },

        # Subnet 2
    "vm16" = {  name = "Assembly", id = 540, node = "pve24", .....
                description = "Assembly Line 3",
                ..... network_device = {bridge = "PNet32"} },
}
```

Si può notare, come chiarito per il codice della rete, che nelle tre zone Production sono presenti quattro macchine virtuali, tre delle quali sono assegnate ad una subnet, quindi a una VNet dedicata, e la rimanente è assegnata ad un'altra subnet diversa, quindi un'altra VNet.
Tutte e quattro le macchine sono comunque appartenenti alla stessa zona Production.

È anche possibile notare dal codice dell'infrastruttura di rete che in realtà le zone 3 e 4 sono posizionate entrambe su uno stesso nodo. Analogamente, le macchine virtuali della "Production Zone 1" e della "Production Zone 2" sono anch'esse posizionate sullo stesso nodo "pve23". Il motivo è dovuto al fatto che nella fase di test si disponeva solamente di quattro nodi, e non cinque come descritto in precedenza. Tuttavia, l'infrastruttura descritta a livello teorico prevede comunque cinque nodi seguendo sempre il concetto di micro-segmentazione.

4.3 Implementazione delle Regole Firewall

Questa sezione si occupa interamente della presentazione dei firewall all'interno di Proxmox, come sono gestiti e come si possono configurare sia manualmente che tramite l'utilizzo di API, in questo caso utilizzando il linguaggio Terraform, per poter garantire una corretta sicurezza di rete.

Come illustrato in precedenza, nelle sottosezioni 4.2.3 e in 4.2.4 sono state applicate delle regole firewall all'interno del cluster Proxmox in modo da poter garantire le corrette implementazioni dei "conduits" tra le diverse zone. Queste regole sono definite attraverso l'interfaccia web di Proxmox che poi le tradurrà in regole del tool "IPTables" all'interno dei nodi specificati.

In particolare, il firewall in Proxmox è gestito come segue [38]. esistono tre diverse direzioni del traffico che possono essere scelte per la definizione delle regole:

In: indica il traffico "incoming", cioè entrante, verso una zona.

Out: indica il traffico "outgoing", cioè uscente, da una zona.

Forward: indica il traffico che sta passando attraverso una zona, originario da una zona diversa e diretto verso una ulteriore.

Ogni regola deve contenere un'azione da eseguire:

ACCEPT: indica che la regola definita per un determinato traffico accetta la connessione.

DROP: indica che la regola definita scarta o rifiuta la connessione senza notificare il mittente.

REJECT: indica che la regola definita rifiuta la connessione inviando una notifica al mittente.

Le regole del firewall possono essere definite per tre zone diverse:

Host: indica uno specifico nodo del cluster a cui applicare le regole definite. Viene identificato attraverso l'uso del nome del nodo, ad esempio "pve22" o "pve24".

VM / CT: indica una macchina virtuale oppure un container definiti all'interno di un nodo. In questo caso è possibile definire solamente il traffico di tipo "incoming" oppure "outgoing", e non "forwarding" cioè quello di inoltro.

VNet : indica una specifica VNet della SDN Proxmox, creata in precedenza. Dal momento che il traffico di una VNet è sempre di tipo "Forward", allora le regole create per i firewall delle VNet avranno solamente direzione "forwarding".

È anche possibile definire le regole a livello di Cluster / Datacenter, in questo modo le regole vengono automaticamente distribuite a tutti i nodi presenti. Tutte le regole vengono memorizzate all'interno di file nel "cluster file system" di Proxmox che è accessibile da tutti i nodi. Infine, il servizio di Proxmox "pve-firewall", presente in ogni nodo, aggiorna le regole IPTables secondo ciò che è definito nel file. Questa modalità di definizione "Cluster level" delle regole è molto comoda, siccome può definire delle policy di default una sola volta per tutti i nodi sottostanti.

Poichè la prima implementazione non presenta alcun tipo di segmentazione interna, essa non definisce alcuna regola del firewall, pertanto questa sezione si concentra sulla presentazione delle restanti due implementazioni, ovvero la "Segmentazione Intermedia" e la "Micro-Segmentazione".

Ogni zona, rappresentata da un nodo del cluster, contiene un firewall di perimetro. Pertanto, la corretta implementazione dei conduits tra le varie zone consiste nel definire delle regole per i firewall perimetrali che consentono esclusivamente il passaggio a determinati tipi di traffico inter-zona. Successivamente, viene definita una policy di "default DROP/DENY" a livello cluster per tutti i tipi di traffico non specificati. In questo modo, tutti i nodi otterranno automaticamente le regole per rifiutare il traffico di tipo "In", "Out" e "Forward" rispetto alle altre zone.

Di seguito, come nelle sottosezioni precedenti, viene presentato il codice Terraform della struttura generica, posto all'interno del file "sdn_resources.tf", per la risorsa del firewall:

```
resource "proxmox_virtual_environment_firewall_rules" "node_level_firewall" {
    for_each = var.firewall

    node_name = each.value.node

    dynamic "rule" {
        for_each = each.value.rules

        content {
            enabled  = rule.value.enabled
            type     = rule.value.type
            action   = rule.value.action
            iface    = lookup(rule.value, "interface", null)
            proto    = lookup(rule.value, "protocol", null)
            source   = rule.value.source
            sport    = lookup(rule.value, "sport", null)
            dest     = rule.value.dest
            dport    = lookup(rule.value, "dport", null)
            log      = lookup(rule.value, "log", null)
            comment  = lookup(rule.value, "comment", null)
        }
    }
    depends_on = [
        proxmox_virtual_environment_vm.test
    ]
}
```

La risorsa appena mostrata, definita dal provider BPG, può essere utilizzata per la creazione di regole del firewall a livello di nodo, se viene compilato il campo "node_name" con il nome di un nodo del cluster, a livello di macchina virtuale, se viene compilato il campo "vm_id" non presente in questo codice, oppure a livello di container, nel caso si compili il campo "container_id" anch'esso non presente nel codice. Se tutti e tre i campi vengono omessi, in quanto opzionali, allora la regola viene applicata a livello cluster.

Come è inoltre possibile notare, all'interno della risorsa sono presenti due "for_each". Il primo viene utilizzato per l'iterazione delle variabili per poter istanziare la risorsa, esattamente come tutte le altre risorse precedenti, mentre il secondo è stato necessario poichè il provider BPG definisce il campo "rule" all'interno della risorsa come una lista di regole.

Essenzialmente, è quindi possibile creare una risorsa per uno specifico nodo, VM, container oppure per il cluster stesso e definire al suo interno direttamente una lista di regole, senza quindi dover creare una risorsa per ogni regola del firewall.

Pertanto, per questa specifica implementazione, è stata presa la decisione di creare

una variabile contenente una mappa di oggetti, esattamente come nelle precedenti implementazioni, che al suo interno contiene una lista di oggetti che rappresenta appunto il campo "rule". Dopodichè, all'interno della risorsa il campo "rule" viene specificato come "dynamic" indicando a Terraform di generare molteplici blocchi dinamicamente in base a un determinato insieme di valori, che sono appunto definiti nella variabile.

Infine, viene inserita la risorsa della macchina virtuale tra le dipendenze, poichè nelle regole può essere specificata direttamente una macchina virtuale come sorgente o destinazione del traffico, definendo il suo indirizzo IP. Per questo motivo, la risorsa del firewall deve essere creata dopo la generazione di tutte le risorse che costituiscono le macchine virtuali per evitare problemi di dipendenza tra risorse, come descritto nella sottosezione 4.1.2.

È presente anche un'ulteriore risorsa che permette di impostare le policy di default a livello cluster, come anticipato prima, in modo tale da bloccare tutto il traffico non espressamente consentito:

```
resource "proxmox_virtual_environment_cluster_firewall" "cluster_default" {
    enabled = true

    # policy di default applicate a tutti i nodi del cluster
    forward_policy = "DROP"
    input_policy = "DROP"
    output_policy = "DROP"
}
```

Questa risorsa è stata creata in modo "tradizionale", essendo l'unica risorsa di questo tipo a dover essere creata.

4.3.1 Configurazione del Firewall per la Segmentazione Intermedia

Come illustrato nella sottosezione 4.2.3, questa implementazione definisce tre zone, alle quali sono assegnate delle subnet e delle macchine virtuali a ciascuna zona.

Le regole dei firewall per questa implementazione sono definite specificando quattro tipi diversi di traffico inter-zona consentito:

- Il primo tipo consente la connessione dal "Data Historian" presente nel nodo 23 (zona 2) verso il "Data Historian Replica" posizionato sul nodo 22 (zona 1).

- Il secondo permette il traffico dal "Data Historian" verso la macchina "SCADA Server" dispiegata sul nodo 24 (zona 3)

- Il terzo tipo di traffico permette la connessione dal "Remote Access Jump Host" nel nodo 22 alla macchina "HMI Tablet" posizionata nel nodo 24. Per questo tipo di traffico viene definita un'ulteriore regola nel firewall del nodo 23, che corrisponde alla zona posizionata nel mezzo, la quale permette l'inoltro del traffico dal nodo 22 al nodo 24

- Infine, il quarto e ultimo tipo di traffico consente la connessione da "Engineering Workstation" nel nodo 23 alla macchina "Laptop Operator" presente nel nodo 24.

Di seguito, il codice, definito analogamente alle altre implementazioni all'interno del file "terraform.tfvars", che definisce la variabile "firewall" contenente gli oggetti che istanziano la risorsa descritta precedentemente:

```
firewall = {
    "pve22" = {
        node = "pve22",
        rules = [
            { enabled = true, type = "out", action = "ACCEPT",
              interface = "enp1s0f0", source = "+sdn/guest-ipam-140",
              dest = "+sdn/guest-ipam-330", log = "info",
              comment = "egress da RA Jump Host a HMI Tablet (pve24)" },

            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-210", dest = "+sdn/guest-ipam-110", ...
              comment = "ingress da Historian DB (pve23) a Historian DB Replica" },
        ]
    }

    "pve23" = {
        node = "pve23",
        rules = [
            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-210", dest = "+sdn/guest-ipam-110", ...
              comment = "egress da Historian DB a Historian DB Replica (pve22)" },

            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-310", dest = "+sdn/guest-ipam-210", ...
              comment = "ingress da SCADA Server (pve24) a Historian DB" },

            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-320", ...
              comment = "egress da Engineering Work a Laptop Operator (pve24)" },

            { ... type = "forward", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-140", dest = "+sdn/guest-ipam-330", ...
```

```
            comment = "forwarding da RA Jump Host (pve22) a HMI Tablet (pve24)" },
        ]
    }

    "pve24" = {
        node = "pve24",
        rules = [
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-140", dest = "+sdn/guest-ipam-330", ...
              comment = "ingress da RA Jump Host (pve22) a HMI Tablet" },

            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-320", ...
              comment = "ingress da Engineering Work (pve23) a Laptop Operator" },

            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-310", dest = "+sdn/guest-ipam-210", ...
              comment = "egress da SCADA Server a Historian DB (pve23)" },
        ]
    }
}
```

Le regole definite nel codice mostrato sopra descrivono esattamente i tipi di traffico permessi, illustrati all'inizio di questa sottosezione. È possibile notare come a ogni tipo di traffico siano associate due regole in due diversi nodi. Questo è necessario perchè ogni tipo diverso di traffico comprende due firewall perimetrali, il primo relativo alla zona da cui esce il traffico, che comprende tutte le regole con il campo "type = out", e il secondo relativo alla zona a cui il traffico è destinato, che comprende tutte le regole con il campo "type = in". Un tipo particolare di traffico, invece, comprende anche una terza regola che permette il forwarding del traffico dalla zona 1 alla zona 3, passando per la zona 2 posizionata nel mezzo.

I campi "source" e "dest" contengono gli indirizzi ip delle macchine virtuali che in Proxmox possono essere riferiti come "+sdn/guest-ipam-140". In questo modo Proxmox identifica una determinata macchina virtuale o container tramite il suo id, in questo caso "140", e automaticamente recupera l'indirizzo IP associato dall'IPAM. È anche possibile definire non solo dei singoli "endpoint" come le macchine virtuali o i container, ma anche riferirsi ad intere sottoreti, utilizzando ad esempio l'istanza "+sdn/idmzNet-all" che in questo esempio indica tutto ciò che è associato alla VNet "idmzNet".

4.3.2 Configurazione del Firewall per la Micro-Segmentazione

Questa implementazione differisce di poco rispetto alla "Segmentazione Intermedia". Infatti, nella "Micro-Segmentazione" viene semplicemente aggiunta un'ulteriore segmentazione di rete, dividendo la zona "Productions" (zona 3) in tre zone differenti. Pertanto, i tipi di traffico consentiti in uscita e in entrata alla zona Productions nella precedente implementazione, verranno moltiplicati per tutte e tre le zone indipendenti.

In particolare:

- Il traffico che in precedenza connetteva il "Data Historian" (zona 1) allo "SCADA Server" (zona 3), verrà suddiviso in tre tipi differenti, ognuno originario della macchina "Data Historian" e destinato, tuttavia, a tre diverse macchine "SCADA Server" posizionate nelle tre diverse mini zone Productions.

- Il traffico che inizialmente veniva generato dalla macchina "Remote Access Jump Host" (zona 1) e destinato alla macchina "HMI Tablet" (zona 3), verrà modificato in tre tipi diversi di traffico, originari allo stesso modo del nodo 22 e destinati alle tre macchine "HMI Tablet" posizionate rispettivamente sui nodi 23, 24 e 25. Oltretutto, per questi tipi di traffico, esattamente come per l'implementazione precedente, è necessario definire tre ulteriori regole per il forwarding all'interno della zona posizionata nel mezzo, ovvero la zona "Operations" (zona 2).

- Infine, il traffico della precedente implementazione che permetteva la connessione dalla macchina "Engineering Workstation" (zona 2) alla macchina "Laptop Operator" (zona 3), verrà definito come tre connessioni differenti originarie del nodo 22 e destinate alle tre macchine "Laptop Operator" posizionate anch'esse sulle tre nuove zone Productions.

Per questo motivo, aumenta il numero di regole da implementare, poichè, come appena descritto, sono presenti due firewall di perimetro aggiuntivi rispetto alla precedente implementazione.

Di seguito viene mostrato il codice per la "Segmentazione Intermedia" modificato e adattato per il tipo di segmentazione descritto in questa sottosezione:

```
firewall = {
  "pve21" = {
        node = "pve21",
        rules = [
            # Type 1 - IN
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-210", dest = "+sdn/guest-ipam-110", ...
              comment = "ingress da Historian DB (pve22) a Historian DB Replica" },
            # Type 2 - OUT
            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-330", ...
              comment = "egress da RA Jump Host a HMI Tablet (pve23)" },
            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-430", ...
              comment = "egress da RA Jump Host a HMI Tablet (pve24)" },
            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-530", ...
              comment = "egress da RA Jump Host a HMI Tablet (pve25)" },
        ]
  }
  "pve22" = {
        node = "pve22",
        rules = [
            # Type 1 - OUT
            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-210", dest = "+sdn/guest-ipam-110", ...
              comment = "egress da Historian DB a Historian DB Replica (pve21)" },
            # Type 2 - FORWARD
            { ... type = "forward", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-330", ...
              comment = "forwarding da RA Jump Host (pve21) a HMI Tablet (pve23)" },
            { ... type = "forward", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-430", ...
              comment = "forwarding da RA Jump Host (pve21) a HMI Tablet (pve24)" },
            { ... type = "forward", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-530", ...
              comment = "forwarding da RA Jump Host (pve21) a HMI Tablet (pve25)" },
            # Type 3 - IN
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-310", dest = "+sdn/guest-ipam-210", ...
              comment = "ingress da SCADA Server (pve23) a Historian DB" },
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-410", dest = "+sdn/guest-ipam-210", ...
              comment = "ingress da SCADA Server (pve24) a Historian DB" },
            { ... type = "in", action = "ACCEPT", ...
```

```
            source = "+sdn/guest-ipam-510", dest = "+sdn/guest-ipam-210", ...
            comment = "ingress da SCADA Server (pve25) a Historian DB" },
        # Type 4 - OUT
        { ... type = "out", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-320", ...
            comment = "egress da Engineering Work a Laptop Operator (pve23)" },
        { ... type = "out", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-420", ...
            comment = "egress da Engineering Work a Laptop Operator (pve24)" },
        { ... type = "out", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-520", ...
            comment = "egress da Engineering Work a Laptop Operator (pve25)" },
    ]
}
"pve23" = {
    node = "pve23",
    rules = [
        # Type 2 - IN
        { ... type = "in", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-330", ...
            comment = "ingress da RA Jump Host (pve21) a HMI Tablet" },
        # Type 3 - OUT
        { ... type = "out", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-310", dest = "+sdn/guest-ipam-210", ...
            comment = "egress da SCADA Server a Historian DB (pve22)" },
        # Type 4 - IN
        { ... type = "in", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-320", ...
            comment = "ingress da Engineering Work (pve22) a Laptop Operator" },
    ]
}
"pve24" = {
    node = "pve24",
    rules = [
        # Type 2 - IN
        { ... type = "in", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-430", ...
            comment = "ingress da RA Jump Host (pve21) a HMI Tablet" },
        # Type 3 - OUT
        { ... type = "out", action = "ACCEPT", ...
            source = "+sdn/guest-ipam-410", dest = "+sdn/guest-ipam-210", ...
            comment = "egress da SCADA Server a Historian DB (pve22)" },
```

```
            # Type 4 - IN
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-420", ...
              comment = "ingress da Engineering Work (pve22) a Laptop Operator" },
        ]
}
"pve25" = {
        node = "pve25",
        rules = [
            # Type 2 - IN
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-120", dest = "+sdn/guest-ipam-530", ...
              comment = "ingress da RA Jump Host (pve21) a HMI Tablet" },
            # Type 3 - OUT
            { ... type = "out", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-510", dest = "+sdn/guest-ipam-210", ...
              comment = "egress da SCADA Server a Historian DB (pve22)" },
            # Type 4 - IN
            { ... type = "in", action = "ACCEPT", ...
              source = "+sdn/guest-ipam-220", dest = "+sdn/guest-ipam-520", ...
              comment = "ingress da Engineering Work (pve22) a Laptop Operator" },
        ]
}
```

Come descritto in precedenza, in questa implementazione vengono aggiunte nuove regole per gestire l'incremento del numero di nodi e, di conseguenza, anche del numero di firewall. Infatti, nel nodo "21" sono presenti tre regole diverse per il traffico in output, commentato come "Tipo 2", che è diretto verso i nodi "23", "24" e "25", mentre in precedenza era definita un'unica regola. Lo stesso tipo di traffico viene catturato dalle regole presenti nel nodo "22", che gestiscono l'inoltro dei pacchetti di rete verso i tre nodi specificati sopra. In questo stesso nodo sono inoltre presenti altri due tipi di traffico, in input e in output, rispetto ai nodi delle tre zone di produzione. Infine, negli ultimi tre nodi sono presenti le regole inverse, corrispondenti ai tipi di traffico definiti negli altri due nodi.

Questa implementazione è stata creata per mostrare a livello teorico come dovrebbero essere definite per ogni nodo le regole dei firewall perimetrali, in modo tale da implementare correttamente il concetto di "conduit". Tuttavia, come descritto nella sottosezione 4.2.4, l'implementazione per la micro-segmentazione è stata eseguita con l'utilizzo di quattro nodi piuttosto che cinque effettivi. Questa modifica, ovviamente, affligge anche l'implementazione delle regole dei firewall appena descritte. Di fatto, l'implementazione corretta per la fase di test non prevede l'utilizzo del nodo "pve25", il che, seguendo l'implementazione delle macchine virtuali presentata nella sottosezione 4.2.4, sposta tutte le

regole definite in esso sul nodo “pve23”, essendo il nodo che ospita due zone di produzione.

4.4 Dalle Implementazioni ai Concetti di SDN e IaC per Cybersecurity

Le implementazioni presentate nel Capitolo 4 permettono di osservare in modo pratico e concreto alcuni dei meccanismi e delle implicazioni di sicurezza presentati nel Capitolo 3. In particolare, i casi di studio mostrano come l'utilizzo combinato di Software-Defined Networking e Infrastructure as Code consenta di applicare alcune pratiche di sicurezza tipiche dei due paradigmi. Uno degli aspetti più importanti riguarda l'utilizzo della segmentazione di rete, introdotta nei casi di studio attraverso la definizione delle diverse topologie di rete nell'infrastruttura Proxmox. Come è stato discusso nel Capitolo 3, la segmentazione rappresenta una tecnica molto efficace per limitare la superficie di attacco e ridurre la possibilità del movimento laterale di un attaccante all'interno di una rete. Nei casi implementati, la definizione delle diverse configurazioni di rete tramite Terraform consente di applicare tale principio dichiarando appunto la topologia della rete insieme a tutte le sue componenti tramite codice, evidenziando come l'approccio IaC permetta di definire e versionare le diverse configurazioni di rete che possono essere riprodotte e applicate in base alle necessità. Un altro aspetto rilevante riguarda la centralizzazione della gestione della rete, una caratteristica tipica delle architetture Software-Defined. Come illustrato nel Capitolo 3, questa proprietà può facilitare l'implementazione, la configurazione e il dispiegamento di controlli di sicurezza e il monitoraggio del traffico di rete, consentendo contemporaneamente una visione globale dell'infrastruttura. Nei casi di studio, l'utilizzo della piattaforma Proxmox per la gestione della SDN mostra come la configurazione delle reti virtuali interne possa essere gestita in modo centralizzato dalla sua interfaccia web. Inoltre, la sezione 4.3 mostra come è possibile configurare i firewall all'interno dei nodi Proxmox tramite l'utilizzo di Terraform, quindi del paradigma IaC, definendo il codice necessario per descrivere le regole da applicare ad essi e poterle gestire e visualizzare successivamente in modo centralizzato grazie all'interfaccia web.

Dal punto di vista dell'Infrastructure as Code, l'utilizzo di Terraform come tool di provisioning introduce inoltre alcuni vantaggi legati alla riproducibilità dell'infrastruttura definita e alla tracciabilità delle modifiche che vengono apportate nel tempo. Come discusso nel Capitolo 3, la definizione dell'infrastruttura tramite codice consente di mantenere delle versioni diverse delle configurazioni e di applicare modifiche in modo controllato, riducendo, in questo modo, il rischio di errori manuali e fenomeni di configuration drift, grazie anche alla possibilità di mantenere tali configurazioni in repository centrali. Inoltre, si sottolinea come Terraform stesso favorisca un approccio di tipo immutabile

per la configurazione di un'infrastruttura, cioè che ogni modifica sostanziale che viene definita nel codice, viene rilevata da Terraform e applicata eseguendo il deprovisioning dell'infrastruttura obsoleta e rieseguendo il provisioning della nuova infrastruttura con le modifiche già apportate. Questo approccio favorisce ulteriormente la riduzione del fenomeno del configuration drift.

I casi di studio presentati, tuttavia, non coprono tutti gli aspetti di sicurezza analizzati nel Capitolo 3. Ad esempio, non vengono affrontati in modo diretto alcuni temi come la sicurezza del controller nelle architetture SDN, la protezione delle API Southbound e Northbound o i meccanismi di autenticazione e autorizzazione utilizzati per l'accesso ai componenti di controllo della rete. Analogamente, nel contesto dell'Infrastructure as Code, non vengono esplorate in dettaglio problematiche come la gestione dei segreti all'interno del codice, la sicurezza della supply chain relativa alla distribuzione dei moduli o l'utilizzo di strumenti di analisi del codice per l'individuazione di vulnerabilità nei template che descrivono l'infrastruttura. Questi aspetti rappresentano possibili direzioni per eventuali estensioni del lavoro, in cui i casi di studio potrebbero essere ampliati, ad esempio integrando meccanismi di sicurezza aggiuntivi. Tali lavori futuri vengono discussi nella Sezione 5.2.

Capitolo 5

Conclusioni

Nel corso di questa tesi sono stati analizzati i paradigmi del Software-Defined Networking e dell'Infrastructure as Code, evidenziando le principali caratteristiche e miglioramenti che entrambi comportano, rispettivamente per le reti e per le infrastrutture, e le implicazioni dal punto di vista della sicurezza di infrastrutture moderne. L'adozione di modelli basati su software per la gestione delle infrastrutture consente infatti di superare alcuni limiti delle architetture tradizionali.

L'approccio SDN consente di separare il piano di controllo dal piano dati, introducendo un livello di gestione che facilita il controllo della rete, il monitoraggio, la configurazione e l'applicazione delle policy di rete. Analogamente, il paradigma IaC permette di descrivere e gestire l'infrastruttura tramite configurazioni dichiarative definendo del codice, migliorando la riproducibilità degli ambienti, facilitando la gestione e tracciabilità delle modifiche e riducendo il rischio di errori manuali.

Attraverso l'utilizzo della piattaforma Proxmox e del tool Terraform, sono stati sviluppati dei casi di studio che mostrano come questi paradigmi possano essere integrati per realizzare infrastrutture di rete OT virtuali e gestite tramite codice. Inoltre, i modelli di segmentazione implementati evidenziano come sia possibile applicare principi di sicurezza come l'isolamento dei sistemi e la riduzione della superficie di attacco direttamente nella fase di progettazione dell'infrastruttura, grazie alla caratteristica dichiarativa dell'IaC tramite la definizione di codice.

5.1 Contributi

Il lavoro presentato in questa tesi fornisce diversi contributi relativamente all'analisi e all'applicazione dei paradigmi SDN e IaC, sia nel contesto dei miglioramenti e delle implicazioni rispetto all'utilizzo di tali architetture per infrastrutture moderne, sia nel contesto della sicurezza delle infrastrutture informatiche.

Inizialmente, è stata condotta un'analisi dell'architettura di entrambi i paradigmi, evidenziandone i vantaggi operativi e le implicazioni relative alla sicurezza. In particolare, sono stati esaminati i principali miglioramenti e meccanismi di sicurezza associati alle architetture SDN e le principali caratteristiche che il modello IaC può offrire, come ad esempio il controllo centralizzato, il controllo dinamico del traffico di rete, l'automazione e la riproducibilità delle configurazioni e la riduzione degli errori manuali. Successivamente, sono state presentate le principali problematiche di sicurezza e vulnerabilità, specifiche per entrambi i modelli.

Un secondo contributo si basa sull'integrazione dei due modelli SDN e IaC attraverso l'implementazione di una serie di casi di studio, realizzati attraverso l'utilizzo della piattaforma Proxmox VE e del tool di provisioning Terraform. Queste implementazioni dimostrano concretamente come sia possibile definire e gestire infrastrutture di rete virtuali tramite codice dichiarativo, applicando dei principi di sicurezza come la segmentazione di rete, la configurazione di meccanismi di sicurezza come i firewall e la gestione centralizzata delle configurazioni dell'infrastruttura, mostrando come tali principi e metodologie possano essere implementati tramite codice direttamente nel processo di definizione di un'infrastruttura.

Nel complesso, i risultati mostrano come l'integrazione tra SDN e IaC rappresenti un approccio efficace per la gestione delle infrastrutture di rete, sia sotto l'aspetto della sicurezza, sia sotto l'aspetto della gestione e configurazione, consentendo una maggiore adattabilità delle operazioni e introducendo nuovi strumenti per la gestione delle configurazioni.

5.2 Lavori Futuri

Come evidenziato nella Sezione 4.4, diversi temi di sicurezza discussi nel Capitolo 3 non sono stati integrati direttamente nelle implementazioni dei casi di studio.

In particolare, aspetti come la sicurezza del controller nelle architetture SDN, la protezione delle API Northbound e Southbound, l'utilizzo non vincolante di tecniche crittografiche come il protocollo TLS per le comunicazioni tramite il protocollo OpenFlow, i meccanismi di autenticazione e autorizzazione per l'accesso ai componenti critici

di controllo della rete come gli switch, rappresentano ambiti che potrebbero essere ulteriormente studiati e implementati basandosi sulla struttura e la ricerca della presente tesi. Un ulteriore ampliamento del lavoro, potrebbe riguardare l'estensione della metodologia utilizzata nei casi di studio ad altri scenari basati su infrastrutture SDN. In particolare, il modello di segmentazione incrementale presentato nel capitolo 4 potrebbe essere applicato a diversi contesti, come le infrastrutture cloud o reti aziendali estese e complesse, con lo scopo di analizzare come le tecniche di segmentazione e di isolamento dei sistemi, introducendo potenzialmente anche delle architetture Zero Trust, possano essere implementate tramite strumenti di provisioning IaC. In questo contesto, l'utilizzo del paradigma IaC potrebbe facilitare il dispiegamento e la replica di diverse topologie di rete, abilitando anche il versionamento di esse, permettendo di confrontare le metodologie di segmentazione implementate per analizzare quali strategie di sicurezza sono più efficaci.

5.2.1 Proposta di un Linguaggio per Distributed Architectures as Code

Un possibile sviluppo futuro riguarda l'introduzione di un linguaggio di alto livello in grado di automatizzare ulteriormente la definizione delle infrastrutture descritte tramite Software-Defined Networking e Infrastructure as Code. Attualmente, strumenti di provisioning come Terraform richiedono la scrittura manuale del codice per le configurazioni dell'infrastruttura attraverso linguaggi dichiarativi specifici, nei quali è richiesta la definizione esplicita di tutte le risorse, le dipendenze e i parametri necessari per il provisioning dell'infrastruttura. Per fare un esempio, nelle implementazioni, se si vuole modificare il codice in modo da dispiegare una determinata zona su un certo nodo del cluster piuttosto che un altro, bisogna eseguire manualmente le modifiche al codice, definendo proprio su quale nodo la zona deve essere creata.

Pertanto, sebbene questo approccio introduca diversi benefici, la definizione manuale delle configurazioni può diventare complessa in infrastrutture di grandi dimensioni o in dei contesti in cui sono presenti numerose risorse. Inoltre, strumenti come Terraform non possiedono dei controlli integrati per il codice dichiarativo, in cui ogni configurazione sia conforme alle relative politiche di sicurezza di rete, per applicare tali controlli, infatti, si necessita l'introduzione di meccanismi e ulteriori strumenti terzi che vengano integrati nel processo di sviluppo e definizione dell'infrastruttura.

Un possibile sviluppo potrebbe quindi consistere nella progettazione di un linguaggio in grado di generare automaticamente le configurazioni IaC e SDN in base a specifiche di più alto livello. Relativamente allo scenario ipotizzato, potrebbe essere possibile

descrivere l'architettura desiderata tramite modelli astratti o descrizioni concettuali dell'infrastruttura, sfruttando anche la possibilità di riutilizzo di modelli terzi in modo automatico, lasciando al sistema il compito di generare il codice necessario per il provisioning delle risorse e l'applicazione di impostazioni tali da raggiungere la configurazione desiderata. Inoltre, tale linguaggio potrebbe anche integrare, possibilmente a livello compilativo o interpretativo, eventuali controlli di sicurezza delle configurazioni rispetto alle policy definite, in modo da generare dei template conformi a gli standard. Infine, si potrebbero implementare ulteriori meccanismi di sicurezza per proteggere eventuali segreti, come credenziali o token di autenticazione, definiti direttamente all'interno del codice.

Un approccio di questo tipo potrebbe ridurre ulteriormente il rischio di errori di configurazione, aumentare la produttività nella gestione delle infrastrutture e facilitare l'adozione di IaC e SDN anche in contesti complessi e infrastrutture molto vaste.

5.2.2 Integrazione AI in SDN e IaC per la Sicurezza

Un ulteriore ambito di sviluppo riguarda l'integrazione di tecniche e strumenti di intelligenza artificiale (AI) all'interno delle infrastrutture definite da codice, sia nel contesto delle architetture SDN, sia nella gestione delle configurazioni IaC.

Le architetture Software-Defined Networking, grazie alla presenza di un controller centralizzato e alla caratteristica di gestione globale della rete, rappresentano un contesto molto adatto all'applicazione di strumenti di apprendimento automatico. In questo ambito, modelli AI potrebbero essere utilizzati, ad esempio come applicazioni di alto livello o integrate direttamente nel controller, per migliorare ulteriormente l'analisi dei flussi di traffico della rete, individuare comportamenti anomali e applicare sistemi automatici di rilevamento delle intrusioni, irrobustendo così la sicurezza di una SDN.

Allo stesso modo, tecniche di intelligenza artificiale potrebbero essere integrate nel contesto delle infrastrutture gestite tramite il paradigma IaC, ad esempio, per analizzare automaticamente le configurazioni infrastrutturali e individuare potenziali vulnerabilità o errori di configurazione prima della fase di provisioning. Questo tipo di approccio potrebbe quindi migliorare la sicurezza delle infrastrutture automatizzate tramite codice, riducendo il rischio di configurazioni errate, facilitando l'individuazione preventiva di rischi di sicurezza ed eseguendo controlli automatici relativi alle conformità delle configurazioni rispetto alle policy di sicurezza esistenti.

L'integrazione tra l'automazione delle infrastruttura definite da codice, le architetture di rete Software-Defined e tecniche di intelligenza artificiale, rappresenta quindi una possibile evoluzione per la gestione delle infrastrutture di rete, con l'obiettivo di rendere i sistemi sempre più autonomi, adattabili e sicuri.

Bibliografia

[1] Red Hat. What is infrastructure as code? https://www.redhat.com/it/topics/automation/what-is-infrastructure-as-code-iac, 2022.

[2] Red Hat. What is provisioning? https://www.redhat.com/it/topics/automation/what-is-provisioning, 2023.

[3] Proxmox. Proxmox virtual environment overview. https://www.proxmox.com/en/products/proxmox-virtual-environment/overview.

[4] Red Hat. What is software defined networking? https://www.redhat.com/en/topics/hyperconverged-infrastructure/what-is-software-defined-networking, 2020.

[5] IBM. Control plane vs data plane. https://www.ibm.com/think/topics/control-plane-vs-data-plane.

[6] Wikipedia. Software defined networking. https://en.wikipedia.org/wiki/Software-defined_networking.

[7] Proxmox. Proxmox ve documentation. https://pve.proxmox.com/pve-docs/chapter-pve-intro.html, 2025.

[8] Wikipedia. Proxmox virtual environment. https://en.wikipedia.org/wiki/Proxmox_Virtual_Environment, 2025.

[9] Wikipedia. Hyperconverged infrastructure. https://en.wikipedia.org/wiki/Hyper-converged_infrastructure, 2026.

[10] Proxmox. Proxmox software defined network. https://pve.proxmox.com/wiki/Software-Defined_Network, 2025.

[11] IBM. Infrastructure as code. https://www.ibm.com/it-it/think/topics/infrastructure-as-code.

[12] Amazon Web Services. Benefits of infrastructure as code. https://docs.aws.amazon.com/prescriptive-guidance/latest/choose-iac-tool/benefits.html, 2026.

[13] Pulumi. What is pulumi? https://www.pulumi.com/what-is/what-is-pulumi/.

[14] HashiCorp. Terraform introduction. https://developer.hashicorp.com/terraform/intro.

[15] HashiCorp. Terraform language documentation. https://developer.hashicorp.com/terraform/language.

[16] IBM. What is terraform? https://www.ibm.com/think/topics/terraform.

[17] HashiCorp. Terraform modules. https://developer.hashicorp.com/terraform/language/modules.

[18] HashiCorp. Terraform resources. https://developer.hashicorp.com/terraform/language/resources.

[19] HashiCorp. Terraform providers. https://developer.hashicorp.com/terraform/language/providers.

[20] HashiCorp. Terraform state. https://developer.hashicorp.com/terraform/language/state.

[21] OpenTofu. Opentofu manifesto. https://opentofu.org/manifesto/, 2023.

[22] Spacelift. Opentofu vs terraform. https://spacelift.io/blog/opentofu-vs-terraform, 2026.

[23] Seungwon Shin, Lei Xu, Sungmin Hong, and Guofei Gu. Enhancing network security through software defined networking. *IEEE*, 2013.

[24] Lukasz Jalowski, Michal Zmuda, and Michal Rawski. A survey on moving target defense for networks: A practical view. 2022.

[25] IBM. What is siem? https://www.ibm.com/it-it/think/topics/siem.

[26] Fortinet. Aaa security. https://www.fortinet.com/resources/cyberglossary/aaa-security.

[27] Cisco. Network segmentation. https://www.cisco.com/site/us/en/learn/topics/security/what-is-network-segmentation.html, 2025.

[28] Fortinet. Network segmentation. https://www.fortinet.com/uk/resources/cyberglossary/network-segmentation.

[29] Diego Kreutz, Fernando Ramos, Paulo Verissimo, Christian Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 2015.

[30] Sandra Scott-Hayward, Sriram Natarajan, and Sakir Sezer. A survey of security in software defined networks. *IEEE Communications Surveys and Tutorials*, 2016.

[31] Antoine Verdet, Mohammad Hamdaqa, Luis Da Silva, and Foutse Khomh. Exploring security practices in infrastructure as code: An empirical study.

[32] Michele Chiari, Marco De Pascalis, and Matteo Pradella. Static analysis of infrastructure as code: A survey. 2022.

[33] HashiCorp. Why policy as code. https://www.hashicorp.com/en/blog/why-policy-as-code, 2025.

[34] Cloud Native Computing Foundation. Cloud native security whitepaper. Technical report, 2020.

[35] Palo Alto Networks. Cloud security report. https://www.paloaltonetworks.sg/company/press/2020/palo-alto-networks-report-finds-poor-security-hygiene-leads-to-escalating-cloud-vulnerabilities, 2020.

[36] Franco Callegati, Saverio Giallorenzo, Alessandro Melis, Simone Melloni, Marco Prandini, and Andrea Vannini. Investigating operational technology attacks as code. *Empirical Software Engineering*, 2024.

[37] IBM. What is a digital twin? https://www.ibm.com/it-it/think/topics/digital-twin.

[38] Proxmox. Proxmox firewall documentation. https://pve.proxmox.com/wiki/Firewall, 2025.