České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra telekomunikační techniky
Obor: Komunikační sítě a internet

Dávková konfigurace síťové infrastruktury v datacentru

Batch configuration of data center network infrastructure

Diplomová práce

Vypracoval: Bc. Jakub Novák
Vedoucí práce: Ing. Tomáš Vondra, Ph.D.
Rok: 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Novák Jméno:Jakub Osobní číslo: 483598

Fakulta/ústav: Fakulta elektrotechnická

Zadávající katedra/ústav: Katedra telekomunikační techniky

Studijní program: Elektronika a komunikace

Specializace: Komunikační sítě a internet

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Dávková konfigurace síťové infrastruktury v datacentru

Název diplomové práce anglicky:

Batch Configuration of Data Center Network Infrastructure

Pokyny pro vypracování:

V datacentrech firmy se používá segmentace sítě z důvodů bezpečnosti. Některé datové toky ale nelze z důvodů objemu filtrovat firewallem a proto se používají Access Control Listy na přepínačích. V současné době jsou pravidla spravována ručně, což vede k častým chybám.
1. Analyzujte současný stav a možnosti dávkové konfigurace pro použitá síťová zařízení.
2. Navrhněte vhodné řešení a implementujte skripty, které ze zadaných pravidel ve strojově čitelné formě syntetizují konfiguraci a nahrají ji na síťová zařízení.
3. Otestujte na virtuálních zařízeních několika výrobců a typů v simulačním prostředí.

Seznam doporučené literatury:

[1] Dokumentace k Terraform dostupná na https://developer.hashicorp.com/terraform [on-line]
[2] Dokumentace k Ansible dostupná na https://docs.ansible.com/ [on-line]
[3] Dokumentace k Normir dostupná na https://nornir.readthedocs.io/en/latest/ [on-line]
[4] Dokumentace k Eve NG dostupná na https://www.eve-ng.net/index.php/documentation/ [on-line]

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Tomáš Vondra, Ph.D. katedra počítačových systémů FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: 06.02.2024 Termín odevzdání diplomové práce: 24.05.2024

Platnost zadání diplomové práce: 21.09.2025

Ing. Tomáš Vondra, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantbere na vědomí, že jepovinen vypracovatdiplomovou prácisamostatně, bezcizí pomoci,s výjimkou poskytnutýchkonzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datumpřevzetí zadání Podpisstudenta

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetku použitú literatúru.

V Prahe dňa ....

Bc. Jakub Novák

Poďakovanie

Touto cestou by som sa chcel poďakovať Ing. Tomášovi Vondrovi, Ph.D za jeho ochotu, trpezlivosť a vedenie diplomové práce a Radkovi Dohnálkovi zo spoločnosti EmbedIT za cenné rady pri vypracovávaní praktického riešenia. Taktiež by som chcel poďakovať mojej rodine a priateľom za ich podporu počas celého štúdia ako aj všetkým pedagógom s ktorými som mal možnosť spolupracovať.

Abstrakt

Automatizácia sieťovej infraštruktúry v dátových centrách predstavuje efektívny spôsob správy sietí pomocou moderných nástrojov a technológií. Táto diplomová práca sa zaoberá automatizáciou správy Access Control Listov v dátových centrách pomocou programovacieho jazyku Python a knižnice Nornir. Cieľom práce je zanalyzovať aktuálny stav siete a implementovať riešenie, ktoré zautomatizuje manuálne procesy, čím eventuálne zvýši efektivitu a konzistenciu konfigurácií. V teoretickej časti sú predstavené základné koncepty automatizácie ako je metodika Infrastructure as Code. Praktická časť práce zahŕňa analýzu súčasného stavu a konkrétny návrh, implementáciu a testovanie nového riešenia v simulačnom prostredí.

Kľúčové slová: Automatizácia, Infrastructure as Code, Access Control List, Python, Nornir

Abstract

Automation of network infrastructure in data centers is an efficient way to manage networks using modern tools and technologies. This thesis deals with automating the management of Access Control Lists in data centers using Python programming language and Nornir library. The main objective of the thesis is to analyze the current state of the network and implement a solution that will automate manual processes, eventually increasing the efficiency and consistency of used configurations. In the theoretical part, basic automation concepts such as the Infrastructure as Code methodology are introduced. The practical part of the thesis includes an analysis of the current state of the network and the specific design, implementation and testing of the new solution in a simulation environment.

Key words: Automation, Infrastructure as Code, Access Control List, Python, Nornir

OBSAH

ÚVOD

V súčasnej dobe rapídneho technologického pokroku, digitalizácie a cloudifikácie sa sieťová infraštruktúra stáva kľúčovým pilierom pre fungovanie väčšiny organizácií. S neustálym príchodom nových technológií sa aj úmerne zvyšujú požiadavky kladené na fungovanie sietí. Medzi tieto požiadavky patria napríklad prenosová kapacita siete, latencia, škálovateľnosť, ale aj bezpečnosť. Sieťová automatizácia teda predstavuje nevyhnutný krok v evolúcií sieťového odvetvia, ktorý umožňuje organizáciám nielen efektívnejšie využívať svoje zdroje a posilniť bezpečnosť svojej infraštruktúry ale aj rýchlejšie reagovať na rôznorodé požiadavky. Tieto riešenia by mali umožňovať rýchly provisioning, nenáročnú aktualizáciu a správu konfigurácií. Eventuálne sa ich aplikáciou dosiahne minimalizovať výskyt ľudských chýb, ktorý je v praxi veľmi častý. Metodológie ako Infrastructure as Code a softvérovo definované siete predstavujú inovatívne prístupy, ktoré umožňujú správcom sietí dynamicky spravovať a hromadne konfigurovať sieťové prvky s použitím efektívnych a flexibilných automatizačných skriptov. Tento pokrok otvára nové príležitosti pre zlepšenie bezpečnosti a to redukovaním manuálnych zásahov.

Táto diplomová práca sa zameriava na problematiku automatizovania sietí, konkrétne na analýzu a implementáciu automatizovaných riešení pre správu Access Control Listov v prostredí dátového centra. Hlavným cieľom práce je zanalyzovať súčasný stav siete v spoločnosti EmbedIT a na základe dostupných možností navrhnúť a implementovať dávkovú konfiguráciu pre sieťové zariadenia za účelom zautomatizovania zaužívaných manuálnych procesov. V teoretickej časti práce je cieľom priblížiť dôležitosť automatizácie v sieťovom odvetví, vysvetliť metodiku Infrastructure as Code a popísať použiteľné nástroje. Ďalším bodom teoretickej časti je popísať architektúru používanú v dátových centrách spolu s príslušnými technológiami, ktoré sú dôležité pre pochopenie problému vyskytujúcom sa v dátovom centre firmy. Hlavnou časťou diplomovej práce je praktická časť, ktorá sa zaoberá analýzou súčasného stavu siete v dátovom centre spoločnosti EmbedIT, výberom vhodného nástroja pre implementáciu dávkovej konfigurácie, samotným návrhom riešenia a v neposlednom rade aj otestovaním daného riešenia v simulačnom prostredí.

1. AUTOMATIZÁCIA SIETÍ

Prvá kapitola popisuje ako automatizácia transformuje tradičné manuálne procesy na efektívnejšie, odolnejšie a rýchlejšie softvérové procesy. Ďalej skúma problematiku spojenú s rastúcou komplexnosťou a veľkosťou sietí, ako aj potenciálne riziká manuálnej konfigurácie, ktoré môžu eventuálne viesť k bezpečnostným alebo prevádzkovým problémom. Dôležité je aj poukázanie na efektivitu automatizácie pri zaisťovaní konzistencie konfigurácie, dodržiavaní bezpečnostných politík a prispôsobovaní sa rýchlym zmenám v dynamickom prostredí. Táto kapitola taktiež poskytuje prehľad o moderných technologických princípoch ako sú softvérovo definované siete a Infrastructure as Code, ktoré hrajú kľúčovú úlohu v dnešnom svete automatizácie.

Pojem automatizácia môže byť definovaný ako proces, ktorý zefektívňuje plánovanie, provisioning, konfiguráciu, prevádzku a následnú optimalizáciu celkovej IT infraštruktúry prechodom z manuálnych úloh na opakovateľné softvérové procesy. Hlavnou výhodou automatizácie je skrátenie času, ktorý je potrebný na implementáciu, údržbu a zavádzanie zmien.

1.1. Nedostatky manuálnej konfigurácie

Vo väčšine starších tradičných sietí bolo zvykom, že správcovia sietí spravovali zariadenia pripojovaním sa pomocou protokolu Secure Shell (SSH) zvlášť po jednom na každé zariadenie. Tento opakujúci sa proces v malých sieťach nebol problém, avšak s rastúcou veľkosťou a komplexnosťou sietí, bola i menšia zmena veľkou časovou prekážkou. V prípade väčších inštitúcií sa jedná o rozmanité topológie, ktoré často zahŕňajú desiatky až stovky zariadení na rôznych geografických lokalitách. Tradičné manuálne metódy sú v týchto prípadoch nielen náročné na zrealizovanie, ale aj náchylné na ľudské chyby, čo vedie k potenciálnym výpadkom, bezpečnostným zraniteľnostiam a neefektívnosti. Rovnako aj menšie syntaktické chyby v zápise konfigurácie, môžu zabrať zdĺhavý debugging. Medzi najznámejšie výpadky ktoré boli spôsobené ľudským faktorom patrí porucha systému Amazon Web Services (AWS) v roku 2017. Manuálna konfiguračná chyba spôsobila výpadok obrovského množstva webových stránok, online služieb a aplikácií, ktoré spoliehali na daný systém[1]. Táto manuálna chyba viedla k odstráneniu väčšieho počtu serverov ako bolo zamýšľané a celkovo tento štvorhodinový výpadok spôsobil stratu vo výške viac ako sedem miliárd českých korún.

Tento incident vzorovo poukazuje na potenciálne riziká spojené s manuálnou konfiguráciou.

1.2. Použitie automatizácie

Dynamika komerčného prostredia vrátane rýchleho nasadzovania nových služieb a škálovania infraštruktúry s cieľom naplniť kolísavý dopyt, si vyžaduje agilnejší a efektívnejší prístup k správe sietí[2]. Jeden z hlavných faktorov je škálovateľnosť. To znamená, že vďaka automatizácií je možné spravovať obrovské množstvo zariadení v rôznych geografických lokalitách z jedného centrálneho bodu. Toto umožňuje hromadne zmeniť konfiguráciu siete podľa potreby bez toho, aby sa zvýšila zložitosť obsluhy. Ďalší faktor je konzistentnosť, pri ktorej automatizované procesy zabezpečujú jednotnú aplikáciu konfigurácií v celej sieti. Týmto krokom sa signifikantne znižuje riziko nezrovnalostí, ktoré môžu eventuálne viesť k bezpečnostným alebo prevádzkovým problémom. Konzistentnosť je taktiež kľúčová pre zachovanie súladu s priemyselnými ako aj organizačnými predpismi a normami. V prípade dohľadávania konfiguračných chýb sa čas potrebný na lokalizáciu chyby výrazne zníži pretože zariadenia zdieľajú jednotnú konfiguráciu. Ďalším významným ukazovateľom ako môže byť automatizácia užitočná je zvýšenie úrovne bezpečnosti. Zautomatizované pracovné postupy môžu rýchlo implementovať bezpečnostné politiky a aktualizácie v celej sieti. Týmto postupom sa zlepší celková bezpečnostná situácia, čo zabezpečí okamžité riešenie zraniteľností a dodržiavanie politiky. V neposlednom rade je dôležité zdôrazniť, že použitím automatizácie sa výrazne skráti čas potrebný na provisioning a konfiguráciu sieťových zariadení, od nasadenia nového hardvéru až po aktualizáciu konfigurácie ako reakciu na zmenu v sieti alebo bezpečnostnú hrozbu. Práve rýchlosť a efektívnosť sú v dnešnom rýchlo-meniacom sa prostredí veľmi dôležité, keď čo i len kratšie prestoje môžu spôsobiť značné finančné dôsledky.

Automatizácia siete zohráva kľúčovú úlohu pri optimalizácii výkonu, bezpečnosti a možnosti správy sieťových zariadení, ako sú switche, routre a firewally. Automatizáciou rutinných úloh môžu organizácie výrazne znížiť počet manuálnych chýb, zvýšiť efektivitu a rýchlejšie reagovať na zmeny v sieti. Praktické použitie automatizácie môže vyzerať nasledovne[2][3]:

- **Správa konfigurácie**

Pomocou zautomatizovaných skriptov alebo nástrojov je možné jednoducho nasadiť nové konfigurácie do sieťových zariadení v celej sieti. Prostredníctvom toho sa zabezpečí konzistentnosť, zníži sa počet konfiguračných chýb a zjednoduší sa proces zavádzania nových alebo zmeny existujúcich nastavení. Automatizácia môže napríklad použiť konfiguráciu VLAN vo všetkých switchoch alebo aktualizovať smerovacie protokoly na routroch bez manuálneho zásahu.

- **Zavádzanie bezpečnostných politík**

Automatizácia môže zefektívniť proces aktualizácie pravidiel firewallu, implementácie systému Intrusion Prevention System alebo Intrusion Detection System[4]. Vďaka nim sa zabezpečí, že sieť zostane zabezpečená proti najnovším hrozbám.

- **Škálovanie siete**

V dynamických prostrediach, ako sú cloudové dátové centrá alebo rýchlo rastúce enterprise siete, umožňuje automatizácia rýchly provisioning sieťových zariadení. Zautomatizované pracovné postupy dokážu uviesť do prevádzky nové switche, routre a firewally v minimálnom čase. A zároveň je možné ich nakonfigurovať podľa vopred definovaných šablón a integrovať ich do už existujúcej sieťovej infraštruktúry bez potreby manuálneho nastavovania.

- **Kontrola súladu s predpismi**

Nástroje na automatizáciu siete môžu priebežne monitorovať a vyhodnocovať konfigurácie sieťových prvkov tak, aby sa zabezpečil súlad ako s internými zásadami tak aj s externými predpismi. Automatické kontroly môžu identifikovať konfigurácie, ktoré nie sú v súlade s predpismi, obsahujú neoprávnené zmeny alebo nedostatky v zabezpečení, čo umožní operatíve tieto problémy okamžite riešiť.

- **Monitorovanie výkonu a riešenie problémov**

Automatizované monitorovacie nástroje môžu zhromažďovať údaje o výkone zo sieťových zariadení, analyzovať prevádzku alebo identifikovať iné anomálie ako napríklad bottleneck. Pri identifikácii problému môže zodpovedný automatizačný nástroj

spustiť proces upozornenia alebo dokonca iniciovať preddefinované postupy na riešenie problémov, to všetko s minimálnym zásahom človeka.

- **Disaster recovery**

Pri plánovaní disaster recovery sieťových zariadení zohráva automatizácia kľúčovú úlohu. Automatizované procesy zálohovania a obnovy zabezpečujú, že konfigurácie kritických zariadení možno rýchlo obnoviť po zlyhaní hardvéru alebo iných poruchách. Pomocou nich sa napríklad minimalizuje downtime a urýchli návrat do prevádzky.

- **Aktualizácie softvéru a správa patchov**

Udržiavanie sieťových zariadení aktualizovaných pomocou najnovšieho firmvéru a softvérových patchov je nevyhnutné pre bezpečnosť a výkon. Automatizácia dokáže naplánovať a vykonať aktualizácie alebo patching v potrebných zariadeniach bez akéhokoľvek potrebného ľudského dohľadu.

Úloha automatizácie v správe siete sa rozšírila nad rámec jednoduchej automatizácie úloh a zahŕňa už aj komplexné stratégie ako koncepty Infrastructure as Code (IaC) alebo softvérovo definované siete (SDN).

1.3. Softvérovo definované siete

SDN predstavuje prístup k riadeniu a správe počítačových sietí, ktorý oddeľuje riadiacu rovinu od dátovej a umožňuje dynamické programovanie sieťových funkcií prostredníctvom centralizovaného SDN kontroléra[5]. Tento model poskytuje vysokú mieru flexibility v riadení trafficu medzi zariadeniami a umožňuje aplikáciám priamo komunikovať s kontrolérom cez rozhranie API pre rýchle implementácie zmien v sieti. Oproti tradičným sieťam, SDN zjednodušuje správu siete tým, že premiestňuje komplexné sieťové funkcie do softvéru, čím otvára nové možnosti pre efektívne nasadenie a bezpečnostné riadenie sietí.

1.4. Infrastructure as Code

Využitie princípu, v ktorom kód spravuje celú infraštruktúru prináša revolúciu v správe infraštruktúry tým, že kodifikuje a automatizuje procesy provisioningu a správy, ktoré sa zvykli vykonávať manuálne. IaC nástroje pomocou konfiguračných súborov zabezpečujú

jednotné prostredie a centralizovane presadzujú zmeny v infraštruktúre. Vďaka tomu sa znižuje počet chýb pri manuálnom zaobchádzaní a eventuálne sa tak zvyšuje celková prevádzková efektívnosť. IaC podporuje deklaratívne aj imperatívne prístupy, ktoré uľahčujú požadovaný stav infraštruktúry[6]. Táto metodika nielen urýchľuje provisioning a znižuje prevádzkové náklady, ale taktiež zvyšuje konzistenciu konfigurácie a spoľahlivosť infraštruktúry. Má kľúčové použitie v praxi ako napríklad v oblasti cloud computingu, virtualizácie alebo rôznych pipeline continuous integration / continuous delivery (CI/CD). Tieto postupy sú základom metodológie DevOps[7, 8], čo predstavuje integračný prístup vo vývoji softvéru. Jedná sa o spoluprácu medzi oddelením vývoja (development) a IT prevádzkou (operations) s cieľom skrátiť životný cyklus vývoja systému. Celkovo možno zhrnúť, že CI/CD zahŕňa súbor postupov určených na automatizáciu procesov určených pre integráciu a doručenie softvéru, čím eventuálne umožňuje dosiahnuť vyššiu kvalitu a spoľahlivosť produktu.

Praktické využitie správy konfigurácie sa využíva napríklad pre definovanie a nasadzovanie infraštruktúry, ako sú routre, switche, virtuálne stroje, load-balanceri alebo aj samotné servery. Rovnaký zdrojový kód je takto rozdistribuovaný pri každom nasadení. Opakovanou synchronizáciou IaC takisto zabraňuje aj manuálnej konfigurácii a presadzuje tak konzistenciu reprezentovaním požadovaných stavov prostredia. Toto je dosiahnuté pomocou intuitívneho a prehľadného kódu v rôznych formátoch, ako napríklad JSON alebo YAML. Čiže ak je potrebné vykonať zmeny, upravuje sa zdrojové, nie cieľové prostredie. Dôležitým princípom IaC je idempotencia, teda schopnosť danej operácie produkovať vždy rovnaký výsledok. Pri použití príkazu na aplikáciu konfigurácie sa vždy nastaví cieľové prostredie do rovnakej konfigurácie bez ohľadu na počiatočný stav prostredia. Tento prístup presúva pozornosť z tradičného skriptovania, takisto náchylného na chyby, na špecifikovanie iba finálnej podoby konfigurácií. Vplyvom toho sa tak napokon zvýši spoľahlivosť a konzistentnosť systémov. Idempotencia sa dosiahne buď automatickou konfiguráciou existujúceho cieľa, alebo zrušením existujúceho prostredia a opätovným vytvorením nového. Samotný kód možno overiť a otestovať pred jeho aplikáciou, aby sa predišlo bežným problémom pri nasadení do produkcie. Ak je to možné, IaC by mal používať deklaratívne definičné súbory. Definičný súbor opisuje komponenty a konfiguráciu, ktoré prostredie vyžaduje. Túto konfiguráciu je možné dosiahnuť aj inak, súbor môže napríklad definovať požadovanú verziu a konfiguráciu servera, ale nemusí špecifikovať proces inštalácie a konfigurácie

servera. Táto abstrakcia umožňuje väčšiu mieru flexibility pri používaní optimalizovaných techník, ktoré dodáva poskytovateľ infraštruktúry. Deklaratívne definície tiež pomáhajú znížiť komplexnosť spojenú s udržiavaním imperatívneho kódu, napríklad skript vykonávajúci provisioning, ktorý môže časom signifikantne narásť. Syntax pre opis IaC zvyčajne závisí od požiadaviek cieľovej platformy. Ako bolo už spomenuté vyššie, platformy najčastejšie podporujú formáty súborov ako YAML alebo JSON[9].

V súčasnom prostredí si zložitosť a objem operácií vyžadujú prechod od manuálnych k automatizovaným procesom v rôznych odvetviach a preto je výber konkrétnych nástrojov kľúčový pre dosiahnutie maximálnej efektivity. IaC nástroje je možné rozdeliť na viacero druhov, pričom každý z týchto rozdielov zohráva kľúčovú úlohu pri vybraní vhodné nástroja pre konkrétnu infraštruktúru. Každý typ predstavuje odlišné metodiky a technológie na správu konfigurácií, čo má vplyv na architektúru systému, bezpečnosť, škálovateľnosť a prevádzkovú zložitosť.

1.4.1. Využitie agenta

Nástroje na správu konfigurácie založené na agentoch fungujú na modeli klient-server. Centrálny server komunikuje s klientskými uzlami a riadi ich prostredníctvom agentov nainštalovaných v každom spravovanom systéme[10]. Agenti na zariadeniach sú zodpovední za vykonávanie príkazov ktoré sú definované a odoslané centrálnym serverom. Mimo toho, agenti zároveň aj predávajú serveru hlásenia o aktuálnom stave klienta a uplatňujú všetky potrebné zmeny podľa požiadaviek centrálneho servera. Pomocou pravidelnej konektivity medzi klientom a serverom je zabezpečené, že konfigurácia klienta zodpovedá požadovanému stavu. Medzi hlavné výhody nástrojov používajúcich agenta patria:

- **Monitorovanie a aktualizácia v reálnom čase**

Nástroje založené na agentoch poskytujú možnosti monitorovania a automatických aktualizácií v reálnom čase. Napríklad úprava bezpečnostných pravidiel, vďaka ktorej je možné udržať sieť aktualizovanú voči vyskytujúcim sa hrozbám. Toto je dosiahnuté tým, že agenti neustále na pravidelných intervaloch hlásia stav svojich hostiteľských systémov centrálnemu serveru.

- **Veľká ponuka rôznych funkcií**

Vďaka prítomnosti agenta tieto nástroje často poskytujú širší rozsah funkcií vrátane podrobnej analýzy systému alebo podrobnejšej kontroly nad konfiguráciami systému.

Nástroje využívajúce agenta prinášajú zároveň aj niekoľko nevýhod:

- **Režijné náklady na zdroje**

Vyššie náklady sú spôsobené tým, že každý agent spotrebúva určité systémové zdroje, čo môže v rozsiahlych prostrediach časom spôsobiť nárast v spotrebe.

- **Zložitosť riadenia**

Nasadenie, aktualizácia a údržba agentov predstavujú v mnohých systémoch prevádzkovú zložitosť.

- **Obavy o bezpečnosť**

Nakoľko každý agent predstavuje potenciálny vektor útoku, musia byť navyše implementované aj preventívne bezpečnostné opatrenia.

- **Nepodporované zariadenia**

V praxi sa bežne v jednej architektúre používajú rôzne typy zariadení a nie všetky musia podporovať inštaláciu agenta na daný operačný systém.

Na druhej strane je dôležité zmieniť, že nástroje na správu konfigurácie bez agentov nevyžadujú inštaláciu ďalšieho softvéru na klientskych uzloch. Zvyčajne využívajú existujúce sieťové protokoly ako napríklad SSH, pre vykonávanie úloh správy na vzdialených systémoch. Takto sa postarajú o zníženie režijných nákladov a o jednoduché nasadenie. V celkovom porovnaní s nástrojmi založenými na agentoch sú ale ich funckionality značne obmedzené.

1.4.2. Fáza projektu kedy sú použité

Použitie nástrojov IaC sa objavuje v rôznych fázach životného cyklu infraštruktúry. Ako výsledok skúmania je možné tvrdiť, že užitočnosť týchto nástrojov primárne závisí od

konkrétnych požiadaviek danej úlohy. Ich použitie je najlepšie rozdeliť do troch hlavných oblastí a to počiatočné nastavenie infraštruktúry (provisioning), priebežná správa infraštruktúry a finálne nasadenie aplikácií s ich následnou správou[11].

- **Počiatočné nastavenie infraštruktúry**

Pri základnom zriadení infraštruktúry sa nástroje vyznačujú deklaratívnou syntaxou a schopnosťou efektívne spravovať komplexné prostredia. Ich výhodou je priame prepojenie s viacerými poskytovateľmi infraštruktúry, ako sú rôzny poskytovatelia cloudových platforiem. Konštrukcia týchto nástrojov uľahčuje vytváranie reprodukovateľnej infraštruktúry a ponúka robustný mechanizmus na počiatočné nastavenie. Príkladom bežne využívaného nástroja v tejto časti je nástroj Terraform[12]. Práve nástroj Terraform obsahuje viacero použiteľných modulov určených na definovanie konkrétnych požiadaviek na cloudových platformách. Jedná sa napríklad o definovanie sieťových nastavení, vrátane rozsahu, subnetov, pravidiel a podobne[13].

- **Priebežné riadenie infraštruktúry**

Následne v oblasti dohľadu a riadenia infraštruktúry sú bežne používanými nástrojmi Ansible alebo Puppet[14]. Tieto nástroje sa vyznačujú idempotentnými operáciami, ktoré zabezpečujú jednoduchú správu stavu a integritu konfigurácie v priebehu času. V praxi toto umožní organizáciám jednorázovo definovať potrebné nastavenia alebo zmeny a následne tieto zmeny jednoducho aplikovať pomocou automatizácie. Tieto nástroje sú schopné správne vykonať zmenu konfigurácie a udržiavať tak súlad celého systému, čo je nevyhnutné pre dlhodobú stabilitu infraštruktúry a konzistenciu konfigurácie.

- **Nasadenie a správa aplikácií**

Pokiaľ ide o nasadzovanie a správu aplikácií, procedurálna povaha systému Ansible umožňuje jemnú kontrolu vďaka čomu je vhodný aj pre zložité architektúry. Vhodným príkladom môže byť aj nástroj Kubernetes, hoci nie je tradičným nástrojom IaC. Kubernetes je príkladom posunu smerom k správe zameranej na aplikácie, ktorá sa skôr zameriava na požadovaný stav ekosystémov aplikácií než výlučne na infraštruktúru[15].

1.4.3. Popis stavu

Deklaratívny prístup je výhodný v tom, že umožňuje používateľovi iba jednoducho špecifikovať konečný stav. Inak povedané, nie potrebné konkretizovať ako presne tento konečný stav dosiahnuť. Nástroje využívajúce deklaratívny model si interpretujú tieto špecifikácie a samostatne určia najefektívnejšiu cestu na dosiahnutie požadovaného konečného stavu. Celkovo možno zhrnúť, že tento model kladie dôraz na jednoduchosť a predvídateľnosť pri správe infraštruktúry.

Naopak, imperatívny model vyžaduje explicitné pokyny na dosiahnutie požadovaného stavu, ktoré popisujú postup krok za krokom. Tento model, ktorý sa často vyskytuje v tradičných postupoch skriptovania a niektorých postupoch správy konfigurácie ponúka podrobnú kontrolu nad procesom nasadenia. To prináša ale aj nevýhodu nakoľko si vyžaduje hlbšie pochopenie základných postupov a môže spôsobiť zložitosť údržby skriptov v priebehu času.

1.4.4. Stálosť

Menný (mutable) systém IaC umožňuje postupné úpravy, čím ponúka flexibilitu a aktualizácie infraštruktúry. Naproti tomu nemenný (immutable) systém považuje infraštruktúru za nemennú a zameriava sa skôr na vytváranie nových zdrojov než na zmenu existujúcich[16].

1.4.5. Master vs Masterless

V architektúrach založených na hlavnom Master serveri, konfiguráciu a správu klientskych uzlov riadi centralizovaný server, ktorý slúži ako autoritatívny zdroj konfiguračných údajov a pravidiel. Tento model uľahčuje centralizované riadenie, ale zavádza potenciálny bottleneck a single point of failure.

Architektúry Masterless odstraňujú závislosť od centrálneho riadiaceho uzla, pričom každý uzol sa dokáže konfigurovať nezávisle, napríklad čerpaním konfigurácie z úložiska. Tento decentralizovaný prístup, ktorého príkladom môže byť nástroj Ansible, zvyšuje škálovateľnosť a odolnosť tým, že distribuuje logiku konfigurácie v rámci celej infraštruktúry.

1.4.6. Push vs Pull

Model Push sa vyznačuje aktívnym prístupom, pri ktorom proces konfigurácie iniciuje spravujúci server. V tomto modeli sa konfigurácie posielajú priamo do cieľových uzlov podľa potreby. Tento prístup uľahčuje okamžité použitie konfigurácií, čo umožňuje rýchle nasadenie a promptne vykonané aktualizácie. Je obzvlášť výhodný v scenároch, ktoré si vyžadujú promptné opravy alebo zmeny, pretože obchádza potrebu uzlov žiadať alebo kontrolovať aktualizácie. Vyžaduje si však priamy sieťový prístup ku všetkým uzlom, čo môže skomplikovať architektúry zahŕňajúce firewally alebo segmentáciu siete.

Na druhej strane, model Pull funguje na pasívnom princípe, keď cieľové uzly pravidelne iniciujú kontakt s centrálnym serverom, aby skontrolovali a stiahli si nové alebo aktualizované konfigurácie. Tento model ponúka uzlom autonómnejší rámec, vhodný do prostredí, v ktorých sú uzly dynamicky spravované a preto nemusia byť neustále prístupné. Model pull zlepšuje škálovateľnosť tým, že rozdeľuje záťaž v sieti, keďže uzly môžu nezávisle, v rôznych časoch aktualizovať svoje konfigurácie. Tento model je obzvlášť vhodný pre prostredia, ktoré uprednostňujú škálovateľnosť a flexibilitu, ako sú napríklad cloudové alebo kontajnerové nasadenia. Okrem toho, niektoré pokročilé nástroje na správu konfigurácie a IaC ponúkajú flexibilitu na prevádzku v oboch režimoch alebo dokonca kombinujú aspekty oboch. Prostredníctvom nich poskytujú hybridný prístup, ktorý možno prispôsobiť jedinečným požiadavkám každého prostredia nasadenia.

2. PREHĽAD NÁSTROJOV

Druhá kapitola sa venuje prezentácii a analýze nástrojov, ktoré umožňujú automatizáciu sieťovej infraštruktúry. Kapitola popisuje rôzne nástroje a platformy, ktoré zefektívňujú nasadenie, konfiguráciu a následnú správu sieťových zariadení v prostredí dátových centier. Rozobraté sú charakteristiky, aplikácie a výhody nástrojov Ansible, Terraform a jazyka Python pre hromadnú konfiguráciu a jeho vplyv na automatizáciu sietí, ako aj konkrétne knižnice a frameworky, ako sú Nornir, Netmiko a NAPALM. Zámerom tejto kapitoly je poskytnúť komplexný prehľad o dostupných nástrojoch pre automatizáciu a ich aplikácií v rôznych sieťových scenároch, od jednoduchého nasadenia konfigurácií až po komplexnejšiu orchestráciu a správu.

2.1. Ansible

Ansible je populárny open-source nástroj, ktorý sa používa v oblasti automatizácie na rôzne účely, vhodný príkladom z praxe môže byť provisioning, správa konfigurácie alebo orchestrácia. Je napísaný v jazyku Python a pre vstupné dáta používa jazyk YAML. Hlavnými prednosťami Ansible sú jednoduchosť, intuitívnosť, užívateľská prívetivosť a všestranné použitie. Ansible uľahčuje širokú škálu IT postupov v rámci konceptu IaC[17].

Primárny spôsob interakcie s nástrojom Ansible je prostredníctvom jeho Playbookov. Playbooky opisujú požadovaný stav systémov, napríklad určitú konfiguráciu switchov [18]. Ansible potom pracuje na uvedení cieľových systémov do požadovaného stavu bez ohľadu na ich aktuálny stav, pomocou čoho zabezpečuje idempotenciu procesu. Tento prístup nielenže znižuje potenciálne chyby, ale tiež zvyšuje reprodukovateľnosť v rôznych prostrediach.

Architektúra Ansible je mimoriadne jednoduchá a efektívna. Je bezagentová, čiže ako bolo vysvetlené v predchádzajúcej kapitole, nevyžaduje žiaden doplnkový softvér na uzloch, ktoré spravuje. Namiesto toho Ansible využíva protokol SSH na komunikáciu s koncovými uzlami[19]. Pre automatizáciu systémov sa Ansible pripojí k ovládaným uzlom a nahrá vo forme push malé programy, inak povedané Ansible moduly. Ansible tieto moduly spustí a po dokončení ich odstráni. Tieto moduly sú navrhnuté tak, aby boli idempotentné, teda aby vykonali zmeny len v tom prípade, že si ich daný systém

vyžaduje. Ansible takto zabezpečuje konzistentnú konfiguráciu softvéru aj hardvéru v rôznych prostrediach a tým rieši potenciálne nezrovnalosti v konfiguráciách. Ansible automatizuje proces orchestrácie, čiže zjednodušuje proces vytvárania a rušenia prostredí, či už v cloude, on-premise alebo hybridných nastaveniach. Takisto automatizuje aj nastavovanie a overovanie bezpečnostných politík, čím zabezpečuje, aby infraštruktúra spĺňala požadované normy. Príklady implementácie:

- **Zjednodušenie správy infraštruktúry**

Pre príklad je možné použiť scenár správy cloudovej infraštruktúry so stovkami inštancií. Ansible môže automatizovať provisioning týchto inštancií, konfiguráciu sieťových komponentov a nastavenie monitorovacích nástrojov. Všetky tieto úlohy by mohol definovať Playbook, ktorý by zabezpečil, že infraštruktúra bude na konfigurovaná konzistentne a s minimálnym zásahom človeka.

- **Nasadenie aplikácií v distribuovaných prostrediach**

Pri nasadzovaní viacúrovňovej aplikácie môže Ansible spravovať nastavenie každej úrovne, od inštalácie závislostí na webových serveroch cez konfiguráciu databázových serverov až po nasadenie kódu aplikácie. Tento proces zabezpečuje jednotné nasadenie aplikácie, či už vo vývojovom prostredí alebo v produkčnom prostredí, tým sa znižujú riziká nasadenia.

Nasledujúca časť sa zaoberá základnými prvkami, objasňuje ich funkcie, aplikácie a kľúčovú úlohu, ktorú zohrávajú v ekosystéme Ansible. Hlavný súbor Ansible sa nazýva Playbook a obsahuje zoznam postupných operácií známych ako tasky. Playbooky sú hlavným mechanizmom, prostredníctvom ktorého Ansible oznamuje požadované stavy a akcie, ktoré sa majú vykonať na cieľových uzloch. Definujú úlohy a konfigurácie, ktoré sa majú použiť v štruktúrovanej postupnosti. Playbook sa potom môže spustiť na konkrétnych zariadeniach, ich názvy a IP adresy sú uložené v inventári. Hostovia môžu byť tiež organizovaní do skupín, čo umožňuje spustenie Playbooku len na hostoch konkrétnej skupiny. Moduly sú opakovane použiteľné samostatné skripty, ktoré Ansible vykonáva na cieľových uzloch. Moduly môžu vykonávať úlohy od inštalácie softvéru, kopírovania súborov až po správu služieb a používateľov. Tasky sú základnými jednotkami činnosti v rámci playbookov a špecifikujú operácie, ktoré sa majú vykonať

na určených hostoch, s využitím modulov. Rola je ďalšia abstrakcia, ktorá umožňuje organizovať úlohy, programy, súbory, šablóny a premenné do ucelených skupín. Role sa používajú pre uľahčenie opakovaných konfigurácií a tak umožnia zefektívniť používanie playbooku. Pluginy sú malé časti kódu, ktoré rozširujú základné funkcie systému Ansible a umožňujú prispôsobiť správanie systému Ansible bez zmeny základného zdrojového kódu.

```
1  ---
2  - name: Playbook
3    hosts: webservers
4    tasks:
5      - name: Ensure apache is at the latest version
6        yum: httpd
7        state: latest
8      - name: Ensure apache is started
9        serveice:
10         name: httpd
11         state: started
12
```

Obrázok 2-1: Názorná ukážka kódu Ansible

Cieľom tohto playbooku je nainštalovať a spravovať webový server Apache na viacerých hostoch. Na začiatku sa identifikuje a zameriava sa na skupinu hostov *webservers*. Je nastavený na spúšťanie so zvýšenými oprávneniami. Playbook opisuje dve hlavné úlohy: inštaláciu balíka *Apache (httpd)* na najnovšiu verziu pomocou modulu *yum* a zabezpečenie spustenej služby *httpd* pomocou modulu *service*. Ak je *httpd* už v najnovšej verzii a beží, Ansible nebude vykonávať zbytočné zmeny.

2.2. Automatizácia sietí pomocou Pythonu

Automatizácia v sieťovom odvetví sa výrazne rozšírila aj vďaka používaniu jazyka Python, vysokoúrovňového programovacieho jazyka známeho svojou čitateľnosťou, jednoduchosťou a širokým uplatnením v rôznych technologických odvetviach[20]. Postavenie Pythonu ako popredného nástroja v oblasti automatizácie sietí vyplýva z jeho rozsiahleho ekosystému knižníc a frameworkov, medzi ktoré patria napríklad Nornir, Netmiko a Napalm. Tieto python-based nástroje ponúkajú rozsiahly výber funkcií na správu konfigurácie a získavanie prevádzkových údajov. Popularitu jazyka podporuje jeho open-source charakter, ktorý viedol k vývoju množstva nástrojov a modulov

špeciálne prispôsobených pre sieťovú problematiku. Neustále vylepšovanie, široká komunita a kontinuálne aktualizácie knižníc zaisťujú, že Python je jedným z najpoužívanejších programovacích jazykov využívaných pre automatizáciu sietí[21].

2.3. Nornir

Nornir je flexibilný a výkonný automatizačný framework napísaný v jazyku Python a určený na automatizáciu vo veľkých sieťach. Vďaka sile jazyka Python, Nornir môže využívať skripty pre sieťovú automatizáciu s komplexnejšou logikou a lepšími možnosťami prispôsobenia. Jeho architektúra je postavená na pluginoch, ktoré možno ľahko rozšíriť alebo prispôsobiť tak, aby vyhovovali špecifickým požiadavkám siete alebo prostredia[22]. Integráciou s Python knižnicami, ako sú Netmiko a Napalm, Nornir uľahčuje priamu interakciu so sieťovými zariadeniami a umožňuje vykonávať úlohy, ako je správa konfigurácie, overovanie stavu siete a automatizovanie provisioningu. Jedným z hlavných prípadov použitia Norniru je správa konfigurácie. Sieťoví inžinieri môžu zabezpečiť konzistentnosť v celej sieti jednoducho pomocou Norniru a to aplikáciou zmeny konfigurácie do viacerých zariadení súčasne. Táto schopnosť je nevyhnutná na implementáciu nových nastavení, aktualizáciu sieťových konfigurácií alebo efektívne zavádzanie bezpečnostných politík.

Nornir dokáže automatizovať opakujúce sa úlohy, znížiť počet ľudských chýb a zrýchliť čas nasadenia od počiatočného nastavenia zariadení až po aplikáciu komplexných sieťových návrhov. To je obzvlášť cenné v dynamických prostrediach, ako sú dátové centrá alebo cloudové infraštruktúry, nakoľko práve v nich sa často vyžaduje rýchla škálovateľnosť. Hlavnú výhodu Norniru je možné demonštrovať v praktickom scenári a to, keď je potrebné zaviesť jednotnú zmenu konfigurácie v sieti pozostávajúcej zo zariadení od rôznych vendorov. Pomocou Norniru je možné napísať skript v jazyku Python, ktorý definuje požadovaný stav a používa inventár Norniru na identifikáciu zariadení podľa typu, roly alebo akéhokoľvek vlastného atribútu. Skript potom môže prostredníctvom abstrakčnej vrstvy Nornir vykonať príslušné konfiguračné príkazy na každom zariadení bez ohľadu na výrobcu.

Ako open-source projekt vytvorený výlučne v jazyku Python poskytuje používateľsky prívetivé rozhranie na efektívnu automatizáciu sieťových úloh. Nornir rozširuje svoje

funkcie tým, že umožňuje zakomponovanie vlastných modulov. Tým uľahčuje využívanie rozsiahleho ekosystému Python na rôzne úlohy, ako napríklad debugging alebo ladenie, ktoré sú v iných automatizačných nástrojoch, ako je Ansible, mimoriadne náročné. Schopnosť frameworku ukladať do vyrovnávacej pamäte, opätovne používať a automaticky uzatvárať otvorené spojenia zabezpečuje optimálny výkon a správu zdrojov.

```
1  from nornir import InitNornir
2  from nornir_utils.plugins.functions import print_result
3  from nornir_netmiko import netmiko_send_command
4
5  nr = InitNornir()
6  result = nr.run(task=netmiko_send_command, command_string="show arp")
7  print_result(result)
8
```

Obrázok 2-2: Názorná ukážka použitia Norniru

Nornir sa často porovnáva práve s nástrojom Ansible. Na rozdiel od Ansible, ktorý nemá výhody plnohodnotného programovacieho jazyka, Nornir si tieto výhody zachováva vďaka svojmu pythonovskému základu. Ďalej možno dodať, že Ansible Playbooky sú síce považované za user-friendly, ale môžu byť náročnejšie na troubleshooting a údržbu oproti Norniru. Takisto Nornir je aj signifikante rýchlejší ako Ansible, ktorý môže byť pomalý a náročný na zdroje kvôli absencii viacvláknového spracovania[23].

2.4. NAPALM

NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) ponúka štandardizované rozhranie API na správu sieťových zariadení od rôznych dodávateľov. Táto knižnica v jazyku Python uľahčuje správu konfigurácie, monitorovanie zariadení a celkovú interakciu so zariadením. Vďaka svojej jednoduchosti umožňuje automatizáciu komplexných sieťových architektúr. Podporuje avšak iba limitovanú škálu platforiem ako sú, Cisco IOS-XR, IOS, NX-OS, Juniper Junos a Arista EOS[24]. Vďaka open-source komunite je prehľadne zdokumentovaný a ďalej aktualizovaný rôznymi vylepšeniami, ktoré postupne rozširujú spektrum kompatibility[25].

```python
from napalm import get_network_driver

driver = get_network_driver ("driver_name")
device = driver(
    hostname="10.0.0.1",
    username="admin",
    password="*****",
    optional_args={"port": 31000})

device.open()
print(device.get_interface())
```

Obrázok 2-3: Názorná ukážka použitia knižnice NEPALM

2.5. Netmiko

Netmiko, je ďalšia knižnica jazyka Python, ktorá zohráva kľúčovú úlohu v oblasti automatizácie sietí, najmä pri zjednodušovaní pripojení pomocou protokolu SSH k sieťovým zariadeniam[26]. Knižnica Netmiko ponúka jednotné rozhranie pre interakciu s rôznymi sieťovými zariadeniami, zjednodušuje tak proces aplikácie konfigurácií naprieč rôznymi platformami. Štruktúrovaný rozbor výstupu Netmiko ďalej pomáha extrahovať potrebné údaje z výstupu zo sieťových zariadení, čo umožňuje sofistikovanejšie automatizačné a monitorovacie procesy. V kontexte automatizácie siete Netmiko vyniká jednoduchosťou používania, spoľahlivosťou a efektívnosťou, s akou zvláda súbežné pripojenia zariadení, čo je v rozsiahlych heterogénnych sieťach nevyhnutné.

2.6. Terraform

Nástroj Terraform je najznámejší nástroj v oblasti IaC. Bol vyvinutý spoločnosťou HashiCorp a získal si značnú pozornosť v oblasti automatizácie sietí vďaka svojmu deklaratívnemu prístupu, ktorý umožňuje definovať sieťovú infraštruktúru pomocou konfiguračného jazyka HCL[12]. Ako bolo popísané v predošlej kapitole, na rozdiel od tradičných imperatívnych metód programovania, deklaratívna syntax nástroju Terraform špecifikuje to, čo spraviť, než ako to spraviť. V reálnych aplikáciách Terraform zohráva kľúčovú úlohu pri automatizácii provisioningu a škálovania sieťovej infraštruktúry, čo organizáciám umožňuje zavádzať spoľahlivé postupy v rámci odvetvia DevOps. Prínos Terraformu k automatizácii sietí zdôrazňuje jeho vplyv na cloudové siete. Napríklad automatizovanie vytvárania a správy virtuálnych sietí, subnetov alebo aj pravidiel prístupu. Azda najväčšou výhodou Terraformu je jeho kompatibilita s rôznymi

poskytovateľmi služieb alebo cloudových platforiem, ako AWS, Azure alebo Google Cloud. Každý poskytovateľ v Terraforme implementuje API komunikáciu so svojimi službami[27]. To umožňuje Terraformu byť univerzálnym nástrojom na správu rôznych typov infraštruktúr. Plánovacie a prediktívne funkcie nástroja poskytujú prehľad o zmenách pred ich aplikáciou, v dôsledku čoho sa zmierňujú riziká spojené s aktualizáciami siete. Terraform uchováva informácie o aktuálnom stave infraštruktúry v stavových súboroch, ktoré pomáhajú určiť aké zmeny sú potrebné na dosiahnutie požadovaného stavu. Vo všeobecnosti Terraform poskytuje robustný a flexibilný spôsob správy infraštruktúry prostredníctvom deklaratívneho prístupu a silnej integrácie s viacerými dominantnými poskytovateľmi služieb.

3. SIETE V DÁTOVÝCH CENTRÁCH

Tretia kapitola poskytuje prehľad o bežne používaných sieťových zariadeniach pre tieto zložité systémy a popisuje ich role v zabezpečovaní neustálej prevádzky a spoľahlivosti. Ďalej sa zameriava na analýzu hierarchických architektúr v dátových centrách, ich porovnanie s moderným prístupom topológií Spine-Leaf a dôsledky na výkon a škálovateľnosť siete. Okrem toho sa kapitola podrobne venuje aj princípom a praktikám sieťovej segmentácie, kľúčovým konceptom pre zvýšenie bezpečnosti a efektívnosti v sieťových operáciách. V závere kapitoly, je prezentovaná dôležitosť používania Access Control Listov a technika Virtual Routing and Forwarding v kontexte moderných dátových centier ako aj ich úloha pri zabezpečení izolovaných smerovacích tabuliek a efektívneho riadenia prevádzky.

Dátové centrá sú v dnešnej dobe považované za kritickú infraštruktúru, ktorá poháňa súčasnú ekonomiku a poskytuje základový kameň pre širokú škálu služieb, od webhostingu až po rozsiahle cloudové výpočty. Príkladom svetovo najväčších dátových centier sú centrá prevádzkované poskytovateľmi cloudových služieb, ako sú AWS, Microsoft Azure a Google Cloud. V týchto dátových centrách sú umiestnené stovky sieťových prvkov podporujúce v mnohých prípadoch aj stovky tisíc serverov. Je známe, že región AWS-US-East v Severnej Virgínii má jednu z najväčších koncentrácií kapacít dátových centier na svete[28]. Je odhadované, že až tretina svetového trafficu prechádza práve cez tento región[29]. Zo zmienených faktov je možné konštatovať že, vzhľadom na ich nenahraditeľnú úlohu v globálnej infraštruktúre je nevyhnutné aby tieto dátové centrá boli vybavené robustnou sieťovou infraštruktúrou.

3.1. Architektúra dátovych centier

Sieťová infraštruktúra dátového centra je vybavená vysokokapacitnými zariadeniami. Medzi bežné zariadenia používané v dátových centrách patria Cisco Nexus Series, Juniper QFX Series a Arista 10000 Series[30]. Všetky tieto pokročilé zariadenia sú navrhnuté tak, aby zvládli vysokú priepustnosť a pokročilé požiadavky na smerovanie charakteristické konkrétne pre prevádzku dátových centier. Siete dátových centier musia dodržiavať prísne normy na zabezpečenie prevádzkyschopnosti a spoľahlivosti. Často sú konštruované s redundantnými systémami napájania a chladenia, aby sa zachovali služby aktívne v prípade zlyhania hardvéru alebo iných problémov.

Siete týchto dátových centier musia efektívne zvládať prevádzku generovanú nespočetným množstvom služieb a aplikácií. Od bežných sietí sa líšia rozsahom, zložitosťou, spoľahlivosťou a technológiami, ktoré používajú na zvládnutie obrovských prevádzkových požiadaviek. Vzhľadom na hustú koncentráciu serverov a služieb, dátové centrá generujú obrovský objem dát, čo si vyžaduje pokročilé riešenia riadenia prevádzky a automatizácie. Tým pádom automatizácia v dátových centrách je nielen žiaduca, ale aj nevyhnutná na riadenie zložitých interakcií medzi rôznymi sieťovými komponentmi.

Architektúra siete pre dátové centrá je prevažne postavená na hierarchickej topológii, ktorá často pozostáva z core vrstvy, agregačnej vrstvy a prístupovej vrstvy[31]. Tento hierarchický dizajn bol základom podpory enterprise aplikácií a aplikácií dátových centier po celé desaťročia. Jadrová vrstva je zodpovedná za vysokorýchlostné smerovanie paketov medzi lokalitami a internetom. Agregačná vrstva slúži ako sprostredkovateľ a poskytuje služby, ako je napríklad load-balancing a segmentácia VLAN. Prístupová vrstva je miestom, v ktorom sa k sieti pripájajú servery a úložiská. Použitie tejto architektúry nielen ponúka minimálnu latenciu ale zabezpečí aj maximálnu dostupnosť šírky pásma. Konkrétne tieto dva detaily sú rozhodujúce pre služby, ktoré závisia od spracovania údajov v reálnom čase. Alternatívou k tomuto modelu je dvojvrstvová architektúra, ktorá konsoliduje agregačnú a jadrovú vrstvu s cieľom zjednodušiť návrh a znížiť latenciu.

Moderné dátové centrá využívajú zároveň aj topológie typu Spine-Leaf, ktoré zlepšujú škálovateľnosť a redundanciu oproti tradičným trojvrstvovým návrhom[32]. Spine-Leaf umožňuje, aby všetky zariadenia boli od seba vzdialené presne rovnaký počet segmentov, čo zlepšuje konštantnosť výkonu v celej sieti. Prechod od tradičných trojúrovňových architektúr k topológiám Spine-Leaf predstavuje významný vývoj v návrhu siete dátového centra. Pre uvedenie do kontextu, v minulosti sa protokol Spanning Tree Protocol (STP) používal na zabránenie vzniku slučiek v topológiách siete. STP však bol problematický v prostrediach rozsiahlych dátových centier, pretože má tendenciu blokovať redundantné cesty, ktoré by sa inak mohli použiť na zlepšenie redundancie a priepustnosti[32]. Zavedenie architektúry Spine-Leaf prekonáva tieto obmedzenia používaním spojení na úrovni L3 medzi switchmi, čo umožňuje využívať všetky dostupné cesty a poskytuje odolnejšiu a efektívnejšiu sieť.

Porovnanie tradičnej trojvrstvovej a dvojvrstvovej architektúry s topológiou Spine-Leaf odhaľuje významné rozdiely v škálovateľnosti, výkonnosti a odolnosti voči chybám. Tradičné siete so svojím štruktúrovaným hierarchickým usporiadaním ponúkajú predvídateľné a stabilné prostredie. Vďaka dobre pochopeným princípom návrhu sa zvyčajne ľahšie spravujú a monitorujú.

Na druhej strane, architektúra Spine-Leaf, je navrhnutá tak, aby vyhovovala dynamickým a horizontálnym modelom prevádzky moderných dátových centier a najmä zlepšila škálovateľnosť a odolnosť voči chybám. Využívaním priamych ciest medzi ľubovoľnými dvoma koncovými bodmi v sieti minimalizujú topológie Spine-Leaf oneskorenie a eliminujú potrebu STP. Táto vymoženosť tak zlepšuje využitie šírky pásma a skracuje čas potrebný na konvergenciu. Táto konštrukcia je obzvlášť vhodná na zvládnutie obrovského množstva prevádzky generovanej aplikáciami a službami v cloudových a virtualizovaných prostrediach.

Tradičná architektúra aj architektúra Spine-Leaf majú svoje výhody, ich výber často závisí od konkrétnych požiadaviek na sieť, potrieb škálovateľnosti a existujúcej infraštruktúry. Tradičné topológie sa môžu stále uprednostňovať v prostrediach, v ktorých sú modely prevádzky predvídateľné a architektonické zmeny sú obmedzené rozpočtom alebo staršími systémami.

3.2. Princíp segmentácie siete a bezpečnosti

Segmentácia siete je v sieťach dátových centier dôležitým postupom, ktorý zahŕňa rozdelenie siete na menšie samostatné segmenty alebo podsiete. Táto stratégia sa používa z rôznych dôvodov vrátane zvýšenia výkonu, zlepšenia bezpečnosti a uľahčenia dodržiavania regulačných noriem. V praxi je možné sa stretnúť aj s pojmom Demilitarizovaná zóna (DMZ). Je to fyzická alebo logická podsieť, ktorá vystavuje externé služby organizácie nedôveryhodnej sieti, zvyčajne internetu. Avšak zabezpečuje, že vnútorná sieť zostane plnohodnotne zabezpečená[33].

Rozdelením veľkej siete na menšie segmenty môžu organizácie efektívnejšie riadiť tok prevádzky a zabezpečiť, aby citlivé informácie boli prístupné len oprávneným používateľom a systémom. Táto izolácia pomáha zmierniť šírenie potenciálnej hrozby

a výrazne tak znížiť riziko pre zvyšok siete. To znamená, ak je niektorý segment ohrozený, narušenie sa môže obmedziť v rámci tohto segmentu.

Jedným z kľúčových bezpečnostných prvkov segmentácie siete je implementácia Access Control Listov (ACL) a pravidiel na firewalle prispôsobených špecifickým potrebám každého segmentu[26]. Tieto kontrolné mechanizmy zabezpečujú, že do segmentu môže vstupovať alebo z neho vystupovať len schválená prevádzka, v dôsledku čoho sa zvyšuje celková bezpečnosť dátového centra. Okrem toho segmentácia podporuje dodržiavanie rôznych regulačných noriem, ako je napríklad norma PCI DSS (Payment Card Industry Data Security Standard), ktorá nariaďuje oddelenie údajov držiteľov kariet od zvyšku sieťového prostredia[34].

Na segmentáciu siete v dátových centrách sa bežne používajú technológie ako VLAN, Virtual Private Cloud (VPC), Virtual eXtelnsible LAN (VXLAN) a ethernetové siete VPN (EVPN) alebo Virtual Routing and Forwarding(VRF)[35]. Technológia VRF je základným kameňom architektúry moderných sietí v dátových centrách a poskytuje možnosť vytvárať izolované smerovacie tabuľky v rámci jednej fyzickej sieťovej infraštruktúry. Zavedenie automatizácie sietí ďalej uľahčuje dynamickú a automatizovanú segmentáciu siete a tak umožňuje hromadnú správu sieťových segmentov na základe predurčených politík v reálnom čase. Keďže architektúry dátových centier sa naďalej vyvíjajú, úloha segmentácie siete pri ochrane a optimalizácii sieťových zdrojov je čoraz dôležitejšia.

3.3. Access Control Lists

ACL sú základným bezpečnostným mechanizmom v dátových sieťach, ktorý slúži ako primárna obrana pri riadení vstupného a výstupného trafficu[36]. Používa sa taktiež pri nastavovaní politík zabezpečenia siete. ACL umožňujú definovanie pravidiel, ktoré povoľujú alebo zakazujú sieťovú prevádzku na základe rôznych kritérií. Tieto kritéria v praxi predstavujú IP adresy, typy protokolov a čísla portov. Primárnou funkciou ACL je teda filtrovanie trafficu do a zo sieťových rozhraní. Konkrétnejšie, ACL presne špecifikujú ktoré pakety môžu prechádzať cez sieťové rozhrania a ktoré nie. Táto schopnosť je kľúčová pre ochrane citlivých oblastí dátového centra, napríklad segmenty, v ktorých sa nachádza kritická infraštruktúra alebo dôverné údaje. ACL sa aplikujú na

routre, switche a firewally, kde kontrolujú pakety s cieľom určiť, či vyhovujú niektorému zo zadaných pravidiel. Filtrovaním nežiaducej prevádzky pomáhajú ACL chrániť sieť pred neoprávneným prístupom a potenciálnymi útokmi. Ďalšou výhodou ktorú ACL prinášajú je, že obmedzením sieťovej prevádzky iba na nevyhnutnú komunikáciu môžu ACL znížiť nadbytočné zaťaženie sieťových zdrojov, a tým aj zlepšiť výkon. V sieťach dátových centier sa používajú najmä dva typy ACL[37]:

- **Štandardné ACL**

Sú jednoduchšie a kontrolujú prevádzku na základe IP adresy. Zvyčajne sa používajú na povolenie alebo zamietnutie prevádzky z konkrétnych hostov alebo sietí.

- **Rozšírené ACL**

Tieto ACL ponúkajú podrobnejšie riadenie a môžu filtrovať prevádzku nielen na základe IP adries ale aj typov protokolov a čísel portov.

Hoci sú ACL výkonnými nástrojmi na zabezpečenie siete, ich manuálny manažment môže byť náročný. ACL sa môžu stať zložitými a ťažko spravovateľnými s narastajúcim počtom pravidiel, v bežnej praxi jeden ACL môže obsahovať aj viac ako sto pravidiel. Manuálne spravovanie jednotlivých pravidiel markantne zvyšuje riziko nesprávnej konfigurácie, ktorá môže viesť k bezpečnostným zraniteľnostiam alebo prerušeniu sieťových služieb. Z praxe je známe, že efektívna správa ACL si vyžaduje pravidelné revízie a aktualizácie, aby sa zabezpečilo, že kontroly prístupu zostanú relevantné a zosúladené s aktuálnymi bezpečnostnými politikami. Riešením pre elimináciu chyby spôsobenej ľudským faktorom je použitie automatizačných nástrojov, ktoré využívajú prehľadnejší zápis v separátnom dokumente. Pomocou týchto nástrojov je tak možné naraz skontrolovať a aplikovať konfigurácie na viacerých sieťových zariadeniach, vďaka čomu sa dosiahne vyššia miera konzistencie a lepšia prehľadnosť.

3.4. Virtual Routing and Forwarding

VRF umožňuje sieťovému zariadeniu udržiavať súčasne viacero rôznych smerovacích tabuliek. Každá inštancia VRF funguje ako samostatný logický router s vlastnou sadou rozhraní, smerovacou IP tabuľkou a forwarding tabuľkou. Toto oddelenie umožňuje izolovať sieťové služby v rámci toho istého fyzického zariadenia, čo nielen zvyšuje

bezpečnosť, ale aj zjednodušuje riadenie prevádzky. Veľkou výhodou tohto riešenia je aj možné zapojenie viacerých subjektov s rovnakými IP adresami[38].

V typickom prostredí dátového centra sa VRF prakticky používa na izoláciu prevádzky rôznych zákazníkov alebo aplikácií. V prípade aplikácií, napríklad oddelenie vývojových, testovacích a produkčných prostredí v rámci tej istej fyzickej infraštruktúry. Vo väčších sieťach dátových centier alebo pri prepojení viacerých dátových centier sa VRF často používa v spojení s metódou Multi-Protocol Label Switching(MPLS)[39]. VRF funguje na tretej vrstve a na oddelenie a identifikáciu prevádzky v rámci širšej siete môže používať značky, ako sú MPLS labels. MPLS labels fungujú ako tagy a pridávajú sa do paketov, to umožňuje efektívne vyhľadávanie trasy a segregáciu prevádzky. Siete VPN MPLS s podporou VRF sú v praxi bežným prvkom pre telekomunikačných operátorov nakoľko umožňujú vytvárať bezpečné, izolované cesty cez spoločnú sieťovú infraštruktúru. Kľúčové komponenty VRF:

- **Inštancie VRF**

Logické časti v rámci routra, ktoré oddeľujú prevádzku. Každá inštancia má svoju vlastnú smerovaciu tabuľku.

- **Route Distinguisher**

Jedinečný identifikátor pridaný k IP trasám, aby sa rovnaký adresný priestor IP mohol používať v rôznych VRF bez vzájomnej kolízie.

- **Route Target**

Rozširujúci atribút, ktorý riadi import a export trás medzi inštanciami VRF a chrbticovou sieťou.

- **VRF leaking**

VRF leaking sa vzťahuje na riadené zdieľanie trás medzi rôznymi inštanciami VRF[40]. Tento proces zahŕňa konfiguráciu politík importu a exportu trás pomocou Route Targetu. VRF leaking je nevyhnutný pre scenáre, v ktorých musia komunikovať oddelené segmenty siete, napríklad medzi rôznymi oddeleniami v rámci organizácie, ktoré pracujú v izolovaných sieťových prostrediach alebo vyžadujú prístup k spoločným službám. Pri

implementácii VRF je potrebné zváženie budúceho škálovania siete, to zahŕňa plánovanie rastu počtu inštancií VRF a súvisiacich smerovacích a forwardovacích tabuliek. Keďže výpočetný výkon narastá a hardware musí vyhovieť rastúcim požiadavkám bez toho, aby bolo spôsobené oneskorenie alebo bottleneck. Na aktívnu správu a riešenie problémov konfigurácií VRF by sa mali používať nástroje na monitorovanie siete, to zahŕňa sledovanie výkonu jednotlivých inštancií VRF a zabezpečenie efektívneho smerovania prevádzky. Takisto nesprávna konfigurácia môže viesť k bezpečnostným zraniteľnostiam, preto je nevyhnutné zaviesť prísne kontroly prístupu a pravidelne kontrolovať konfigurácie VRF.

4. SÚČASNÝ STAV SIETE

Táto kapitola sa podrobne zaoberá architektúrou a topológiou siete použitej v konkrétnom datacentre spoločnosti EmbedIT. V úvode sú predstavené fundamentálne komponenty infraštruktúry, od jadrovej vrstvy cez prístupovú vrstvu, až po koncové zariadenia. Dôraz je preto kladený na rozbor funkcionalít a integrácie rozličných sieťových prvkov. Predmetom analýzy je stratégia zabezpečujúca vysokú dostupnosť a efektívnu prevádzku, ako aj využitie rôznych technológií poskytujúcich bezpečnosť internej komunikácie. Cieľom kapitoly je poskytnúť holistický prehľad súčasného stavu siete a identifikovať možné oblasti pre jej optimalizáciu.

4.1. Jadrová vrstva

Infraštruktúra v datacentre pozostáva z 2-vrstvovej architektúry, čo prakticky znamená, že jadrová a distribučná vrstva sú zjednotené do jednej. Jadrová vrstva pozostáva z dvoch fyzických switchov, ktoré sú nakonfigurované ako jeden logický celok. Toto nasadenie môžeme nazvať ako Dual-switch jadro a je nakonfigurované v móde High-Availability, čo poskytuje redundanciu, zlepšuje odolnosť voči chybám a zabezpečuje neprerušenú sieťovú prevádzku. Táto logická integrácia jadrovej a distribučnej vrstvy taktiež zefektívňuje prevádzku, redukuje komplexnosť a uľahčuje celkové riadenie a škálovateľnosť. Pre zabezpečenie maximálnej funkčnosti chodu dvoch fyzických switchov ako jeden logický prvok je potrebná vysoká rýchlosť prenosu medzi nimi. Dostatočná rýchlosť je zabezpečená prenosovým spojom s kapacitou 100G, čo je v tomto prípade dostatočná kapacita pre podporu rozsiahleho dátového prenosu medzi zariadeniami.

Označenie jadrových switchov je RSW01 a RSW02. Skratka RSW označuje pojem Router-Switch, čo je možné vysvetliť ako Multi-Layer switch, ktorý operuje jednak na vrstve L2 ako bežný swtich, ale aj na vrstve L3 a umožňuje tak inter-VLAN routing, ktorý bude predmetom záujmu v nasledujúcej kapitole. Jedná sa konkrétne o model Cisco Nexus 9300-FX3 Series[41], ktoré sú navrhnuté pre prostredia dátových centier, aby poskytovali vysoký výkon, nízku latenciu a vysokú hustotu prevádzky. Konkrétne model N9K-C93180YC-FX3 je navrhnutý pre vysokú škálovateľnosť a prevádzkovú flexibilitu, čo z neho robí ideálnu voľbu pre moderné dátové centrá, ktoré sa snažia optimalizovať

svoju sieťovú infraštruktúru pre efektivitu a budúci rast. Tento Cisco switch beží na operačnom systéme NX-OS 10.2(4).

Koncept High-Availabilty je medzi core switchmi zaistený pomocou Cisco protokolu Hot Standby Router Protocol (HSRP). HSRP protokol zabezpečuje redundanciu a je navrhnutý tak, aby umožňoval transparentné prepájanie zariadení a zabezpečoval dostupnosť siete tým, že poskytuje nepretržité preposielanie dát aj v prípade zlyhania aktívneho zariadenia. Pomocou HSRP sú dve alebo viac zariadení nastavené ako skupina pohotovostných zariadení, ale iba jedno je zvolené ako aktívne na preposielanie dát. Ostatné zostávajú v pohotovostnom režime, pripravené prevziať riadenie, ak aktívne zariadenie vypadne. Tento protokol sa bežne používa vo väčších sieťových prostrediach na zabezpečenie vysokej spoľahlivosti siete a zabránenie jedinému bodu zlyhania v infraštruktúre[42].

Celé označenie switchov v dokumentácii je ako hci-purk-rsw01-lan a hci-purk-rsw02-lan, tento detail označenia je dôležitý pre špecifikáciu konkrétneho dátového centra. Označenie HCI symbolizuje región, v tomto prípade sa jedná o hlavnú pobočku v meste Brno a označenie PURK špecifikuje dátové centrum na Purkyňovej ulici. V iných pobočkách sa využíva analogická topológia, avšak s menšími modifikáciami, ako napríklad použitie zariadení od iného vendora.

4.2. Prístupová vrstva

Prístupová vrstva je zložená z dvojíc switchov, ktoré sú taktiež zapojené v móde High-Availability, čiže logicky vystupujú ako jeden celok. Táto vrstva pripojuje koncové uzly, vrátane serverov a úložných jednotiek. Tieto prístupové switche sú prepojené do vyššej vrstvy pomocou 4 linkov, z ktorých 2 idú na každý jadrový switch a sú zapojené pomocou technológie Port Channel. Port Channel, je proprietárny Cisco názov pre EtherChannel, čo je technológia, ktorá umožňuje spojenie viacerých fyzických ethernetových liniek do jednej logickej linky s cieľom zvýšiť šírku pásma a zabezpečiť redundanciu[43]. Agregácia viacerých Port Channel spojení účinne zvyšuje odolnosť spojenia, pretože prevádzka môže pokračovať aj v prípade zlyhania jedného zo združených spojení. Táto technológia optimalizuje využitie sieťových zdrojov a zlepšuje celkovú efektívnosť siete tým, že dokáže rozdeliť sieťovú prevádzku medzi všetky dostupné spojenia v kanáli.

Prepojenie dvojíc switchov s core switchmi je pomocou štyroch 10G linkov, taktiež ako EtherChannel. Access switche sú prepojené medzi sebou ako 2x40G link cez EtherChannel. Zapojenie serverov je pomocou dvoch 10G linkov, jeden link ide na každý z dvojice switchov, tieto spojenia sú taktiež v móde EtherChannel.
Prístupová vrstva pozostáva zo switchov Cisco Nexus 5020 s označením dvojíc nasledovne: NX01/02, NX03/04, NX05/06. Celové označenie teda vyzerá ako hci-purk-nx01-ic. Zariadenie Cisco Nexus 5020, ktoré je súčasťou rady Nexus 5000 Series, je navrhnuté tak, aby poskytovalo prepínanie s vysokou hustotou, primárne určené na nasadenie v dátových centrách ako Top-of-Rack (ToR) switch. ToR zároveň aj podporuje funkcie nevyhnutné pre konvergované sieťové prostredia[44]. Toto zariadenie zabezpečuje efektívnu prevádzku virtuálnych strojov s nízkou latenciou. Týmto optimalizuje prevádzkové požiadavky v prostredí s vysokými výpočtovými nárokmi. ToR swtich je bežný prístup v dátových centrách, keď switch je umiestnený v hornej časti každého racku. Tento dizajn minimalizuje dĺžku káblov a zlepšuje ich správu, prostredníctvom čoho sa nielen znižuje latencia ale aj potenciálne body zlyhania. ToR switche sú zvyčajne prepojené s jadrovými switchmi, aby sa zabezpečila vysoká dostupnosť a výkon[45]. Na danom zariadení beží OS Cisco NX-OS 5.2(1)N1(1b).

Hoci oba modely predstavujú robustné riešenia pre siete dátových centier, séria 9300-FX3 poskytuje platformu, ktorá je odolnejšia a lepšie sa prispôsobuje vyvíjajúcim sa architektúram dátových centier a požiadavkám na automatizáciu. Pre porovnanie, séria switchov Cisco Nexus 9300-FX3, ktorá je použitá v jadre, ponúka pokročilejšie funkcie s vyššou priepustnosťou, lepšou škálovateľnosťou a zvýšenou energetickou účinnosťou. Vďaka zmieneným funkcionalitám tak podporuje širšiu škálu rôznych aplikácií v dátových centrách. Ako bolo zmienené vyššie, veľkou výhodou použitej architektúry je jej jednoduchá škálovateľnosť. Napríklad v prípade expanzie postačuje iba jednoducho pridať nové prístupové switche, ktoré budú obsluhovať nové koncové zariadenia.

4.3. Koncové zariadenia

Ako koncové zariadenia sú použité rôzne typy serverov, ako napríklad ESXi, Blade servery a Bare Metal servery.

- **ESXi**

Virtuálny nástroj ESXi je vyvinutý spoločnosťou VMware a slúži ako hypervisor. ESXi sa inštaluje priamo na fyzický server a tak umožňuje spustenie viacerých virtuálnych strojov na jednom fyzickom serveri. V prostrediach dátových centier sú virtuálne servery kľúčové pre efektívne využívanie zdrojov. Ich zdroje sú špecifikované a upravované podľa aktuálnej potreby, to znamená značné šetrenie nákladov a úsporu energie. Virtuálne servery zároveň poskytujú oproti bežným serverom vyššiu úroveň flexibility, ktorá umožňuje rýchle nasadenie, správu a migráciu virtuálnych strojov[46].

- **Blade servery**

Blade servery sú umiestnené v blade chassis a ponúkajú kompaktné a energeticky úsporné riešenie, ktoré konsoliduje viacero serverov typu blade v jednom racku s cieľom znížiť fyzický priestor a spotrebu energie. Sú ideálne pre škálovateľné výpočtové prostredia s vysokou hustotou a umožňujú jednoduchú integráciu a centralizovanú správu[47].

- **Bare metal servery**

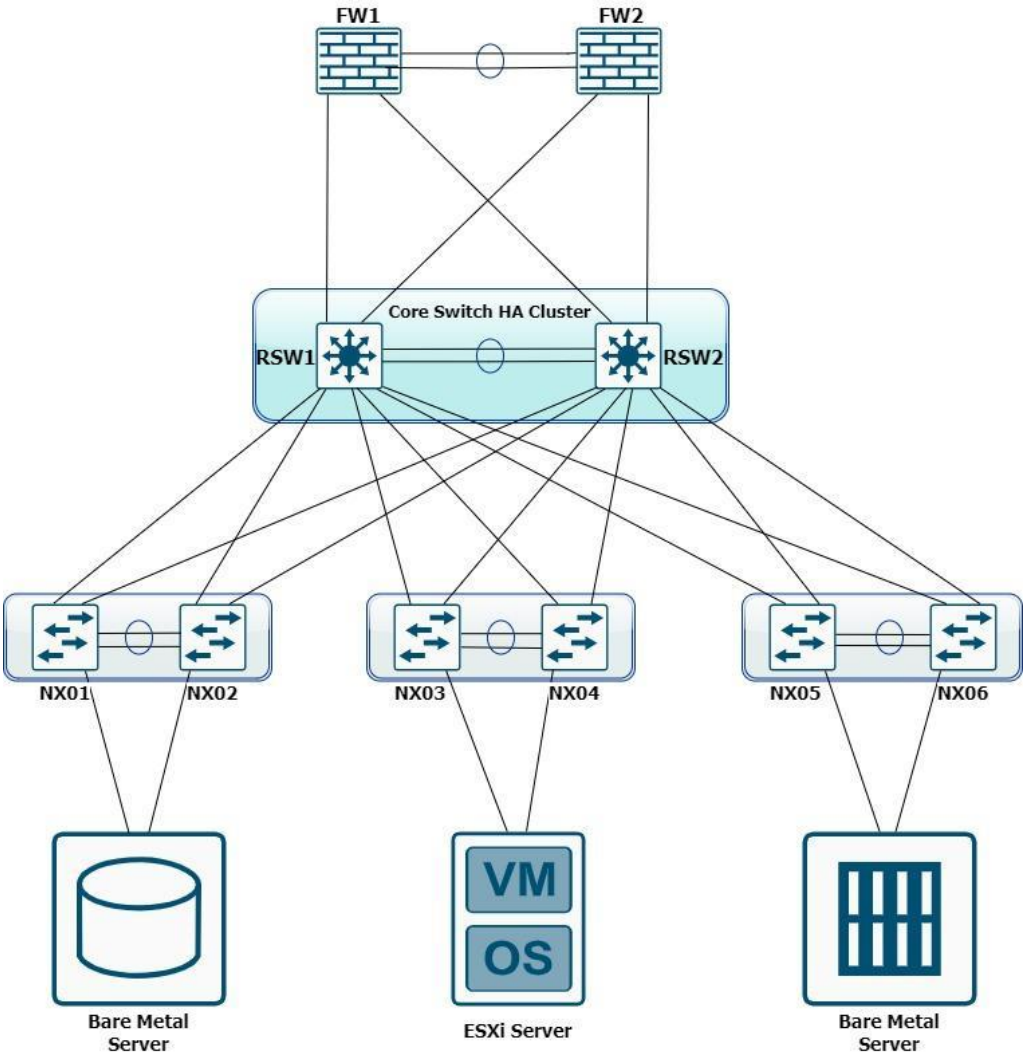
Bare metal servery ponúkajú maximálny výkon tým, že eliminujú réžiu virtualizácie. Tieto servery sa často používajú pre aplikácie vyžadujúce konzistentný vysoký výkon, priamy prístup k hardvérovým zdrojom alebo izoláciu od iných pracovných záťaží[48].

4.4. Firewall

Ďalším prvkom v topológii je firewall. Ako je už zvykom, vo väčšine dátových centier sú použité 2 fyzické firewally, ktoré sú prepojené a fungujú v móde High Availability ako jeden logický celok, inak povedané cluster. V tomto konkrétnom prípade bol použitý Geo-cluster, to znamená, že jeden firewall je umiestnený v datacentre DC1 a druhý v datacentre DC2, a sú medzi sebou prepojené. Je dôležité zmieniť, že dátové centrá sa používajú v konfigurácii Active-Standby. V tomto kontexte práve DC2 slúži ako backup. Toto usporiadanie umožňuje geografickú redundanciu a uľahčuje tzv. disaster recovery, čiže obnovu po havárií alebo výpadku. Unikátnosť tohto pripojenia spočíva v relatívnej blízkosti datacentier, čo v tomto prípade činí zhruba 10km. Pripojenie je redundantné pomocou 2 optických vlákien, s dĺžkou 10 a 21 km. Avšak takéto zapojenie je možné využiť iba v prípade ak vzdialenosť medzi zariadeniami nie je príliš veľká, nakoľko komunikácia v rámci clustru musí spĺňať určité limity na latenciu.

Ako firewall sa používa zariadenie Check Point 16200. Jedná sa o bezpečnostné zariadenie zo série Quantum Security Gateway spoločnosti Check Point, ktoré je

navrhnuté na komplexnú prevenciu hrozieb a správu zabezpečenia pre veľké podnikové siete a dátové centrá[49]. Firewall poskytuje robustný výkon na spracovanie veľkých objemov prevádzky s pokročilými funkciami prevencie hrozieb. Model 16200 je vybavený operačným systémom GAiA R81.10 a ponúka jednotnú bezpečnostnú architektúru, ktorá umožňuje efektívnu správu, monitorovanie a škálovanie bezpečnostných operácií v komplexných sieťových prostrediach. Prepojenie firewallu s jadrovými switchmi je pomocou dvoch 40G spojení.



***Obrázok 4-1*:** Fyzická topológia dátového centra

4.5. Architektúra siete

V moderných dátových centrách sa často používa architektúra typu Leaf-Spine. Ako bolo vysvetlené v predošlej kapitole, toto riešenie eliminuje smyčky na linkovej vrstve ktoré

vznikajú pripojením veľkého množstva zariadení. Toto zapojenie využíva sieťovú vrstvu a je teda očakávané, že koncové zariadenia budú pripojené pomocou sieťovej vrstvy, nie linkovej. Zapojenie na úrovni IP adries je využívané hlavne v cloudových aplikáciách, v ktorých sa využíva kontajnerizácia aplikácií. Avšak v tomto konkrétnom datacentre všetky koncové zariadenia vyžadujú pripojenie iba pomocou VLAN na linkovej úrovni, keďže sa jedná iba o aplikácie pre interné účely. Z praktického hľadiska sa jedná najmä o databázy, ktoré sa využívajú iba lokálne, aby sa predišlo problémom s latenciou. Preto sú všetky potrebné operácie centralizované a routing mimo datacentra tak nie je potrebný.

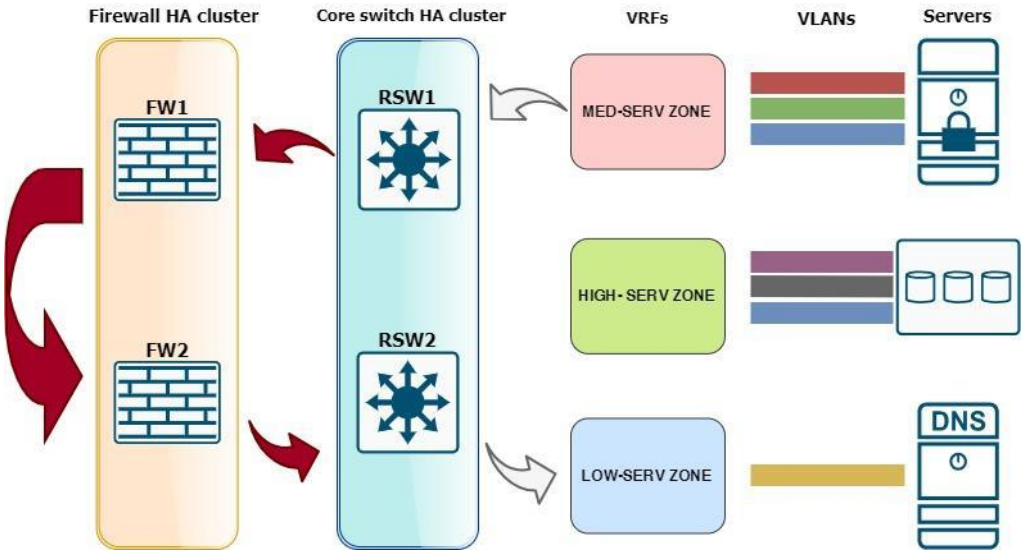
Počas prepínania (switch-over) alebo migrácie do záložného dátového centra sa IP adresy pridelené konkrétnym službám neprenášajú priamo. Namiesto toho sa aplikácie, najmä tie, ktoré sú orchestrované pomocou Kubernetes, presúvajú na úrovni aplikácie, čo vedie len k zmenám DNS bez potreby replikovať pripojenie VLAN alebo konfigurácie IP adries v druhom datacentre. Tento prístup zabezpečuje, že aplikácie nie sú závislé od konkrétnych IP adries alebo konfigurácií VLAN, čo uľahčuje plynulú kontinuitu aplikácií a zjednodušuje správu počas prepínania alebo obnovovaní prevádzky po havárii.

4.6. Použitý routing a switching

Pripojenie serverov pomocou L2 konektivity v rámci tejto siete je navrhnuté tak, aby zaistilo bezpečný a efektívny tok dát. Každý server je priradený k určitej VLAN a prevádzka z týchto sietí je smerovaná do core switchov RSW. V RSW je zvyčajne VLAN ukončená na L3 rozhraní v rámci určenej inštancie VRF, ktorá kontajnerizuje sieťové segmenty na zvýšenie bezpečnosti. V rámci jednej VRF môže koexistovať niekoľko sietí VLAN, to znamená, že je možné vytvoriť odlišné siete pre aplikačné servery. Z hľadiska bezpečnosti sa tieto VRF považujú za bezpečnostné zóny. Inak povedané bezpečnostná zóna reprezentuje určitú VRF a z praktického hľadiska je možné ich používať pre rovnaký účel. V tejto konfigurácii určená VLAN funguje ako vyhradená cesta pre traffic, ktorý sa presúva z VRF na firewall a odtiaľ ďalej do svojho konečného cieľa, či už je to internet alebo iné externé služby. Napríklad server v rámci VLAN 100 má svoju prevádzku smerovanú cez RSW, kde VRF ukončuje rozhranie L3 VLAN 100, a odtiaľ pokračuje do firewallu cez alternatívnu VLAN, napríklad VLAN 101. Táto konfigurácia vymedzuje bezpečnostné zóny a je v súlade so zobrazenou schémou, na ktorej je znázornené smerovanie medzi VRF a zdôraznená rozhodujúca úloha firewallu

pri riadení toku prevádzky. Práve definovanie bezpečnostných zón pomocou VRF je odporúčaná stratégia na zvýšenie bezpečnosti a riadenia prevádzky, ktorú doporučuje aj samotná spoločnosť Cisco.

Konkrétne, v prípade serverov ESXi sa komunikácia doručuje cez trunk porty umožňujúce prenos viacerých VLAN, zatiaľ čo databázové servery využívajú vyhradený access port. V prípade tohto datacentra, sa pôvodne na inter-VLAN routing nepoužívali core switche RSW, ktoré túto funkcionalitu podporujú, ale práve Firewall. Zariadenie Check Point 16200 môže vykonávať routing medzi sieťami VLAN tým, že funguje ako zariadenie na 3. vrstve, ktoré smeruje prevádzku medzi rôznymi sieťami VLAN definovanými v rámci bezpečnostnej politiky. Na identifikáciu a segregáciu sieťovej prevádzky tradične využíva ich označenia. Zároveň uplatňuje bezpečnostné politiky a pravidlá, ktoré riadia prístup a tok prevádzky medzi jednotlivými segmentmi. Táto schopnosť umožňuje firewallu nielen zabezpečiť danú sieť, ale aj riadiť vnútornú prevádzku a účinne kontrolovať prístup medzi rôznymi segmentmi siete.

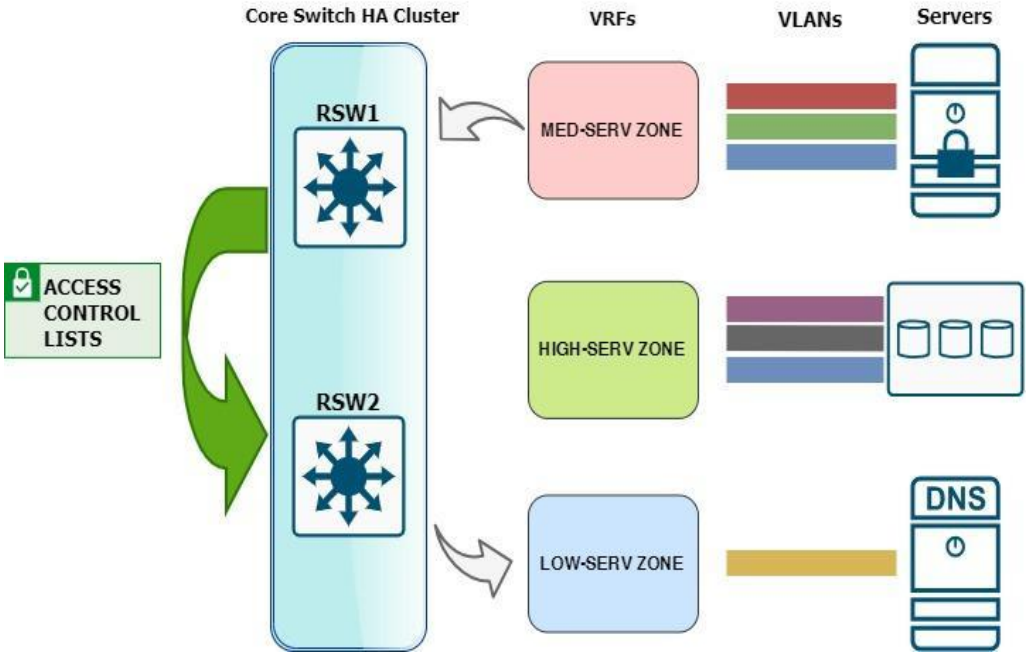


Obrázok 4-2: Stará topológia využívajúca Firewall pre inter-VLAN routing

4.7. Problém riešenia

V použitej architektúre nastáva problém keď Check Point firewall síce podporuje veľký objem dát, ale nie natoľko aby poskytol potrebnú kapacitu, ktorú datacentrum vyžaduje

na veľkoobjemové smerovanie medzi rôznymi VLAN. To je riešené implementáciou VRF leakingu, pomocou ktorého môže sieť efektívne smerovať prevádzku priamo medzi zónami bez preťaženia firewallu. To vedie k zachovaniu jeho zdrojov pre externé bezpečnostné kontroly a prevádzku smerujúcu do internetu. Výhodou použitého VRF leakingu umožňuje prevádzke medzi rôznymi VRF obísť firewall. Práve na základe VRF leakingu sa aditívne zabezpečí, že kľúčové služby zostanú funkčné aj v prípade problémov na firewalle. Táto technika nielenže optimalizuje výkon siete znížením zbytočného zaťaženia firewallu, ale tiež zachováva bezproblémovú komunikáciu medzi VRF, ktorá je kritická pre prevádzku dátového centra.



Obrázok 4-3: Topológia využívajúca VRF leaking

V schéme na obrázku 4-3, je načrtnutá topológia siete využívajúca VRF na riadenie a zabezpečenie toku interných služieb v rámci dátového centra. Dôraz sa kladie najmä na zónu s nízkym stupňom zabezpečenia LOW-SERV. Je nutné zdôrazniť, že táto zóna je v rozpore s tým, čo by mohol naznačovať jej názov. Zóna LOW-SERV je totižto kľúčová pre zachovanie nepretržitého poskytovania základných služieb, ako sú DNS a monitoring. Z praxe je známe, že zrovna tieto služby musia byť ľahko dostupné všetkým serverom v dátovom centre bez toho, aby boli smerované cez firewall. Existujú aj ďalšie vymedzené bezpečnostné zóny vrátane stredne zabezpečenej (MED-SERV),

ktorá sa zvyčajne používa pre neprodukčné, testovacie a vývojové prostredia bez údajov zákazníkov. Posledná zóna je zóna vysokého zabezpečenia (HIGH-SERV), ktorá obsahuje najcitlivejšie informácie, ako sú údaje zákazníkov, a vyžaduje najvyššiu úroveň zabezpečenia a obmedzený prístup. Keďže služby v zóne LOW-SERV sú základom prevádzky dátového centra, prostredníctvom VRF leakingu sa vytvorí cielený bypass. Napríklad zóna LOW-SERV importuje route-targety z iných VRF, ale zo zóny LOW-SERV sa exportujú len vybrané ciele trasy, čím sa zabráni prístupu do všetkých sietí bez rozdielu. Tento selektívny leaking trasy tvorí podstatu stratégie obchádzania firewallu. Do LOW-SERV zóny má prístup teda každá iná zóna, aby mohla pristupovať k daným službám. Tu ale vzniká priestor pre riziko, napríklad pri napadnutí low-serv zóny by sa mohol útočník dostať do každej zo zón, do ktorej má zóna low-serv prístup. Táto potenciálna vulnerabilita je ošetrená práve ACL a povolením trafficu iba pre určité porty. Tento otvorený kanál si teda vyžaduje implementáciu ACL na VLAN L3 rozhraniach pre filtrovanie prevádzky vstupujúcej a vystupujúcej zo zóny LOW-SERV. ACL chránia pred neoprávneným prístupom a zabezpečujú, že traffic prechádza len tam, kam je určený. Tieto ACL sú pre jednoduchosť nakonfigurované v jednom smere, buď prichádzajúcom, alebo odchádzajúcom, aby sa zabránilo neoprávnenému prístupu alebo činnostiam na produkčných serveroch. To ale prináša značnú nevýhodu, pretože veľké množstvo ACL musí byť spravovaných manuálne sieťovým admistrátorom.

5. NÁVRH DOKUMENTÁCIE

Piata kapitola je venovaná popisu návrhu dokumentácie, čo je jeden z kľúčových aspektov diplomovej práce. Dôležitou súčasťou riešenia je návrh hierarchického zápisu použitých zariadení, ich rozhraní a aj jednotlivých ACL. Toto umožní udržiavať prehľadnú a systematickú dokumentáciu, ktorá poslúži ako zdroj pravdy pre použitú sieťovú infraštruktúru. Predpokladá sa, že zavedenie tohto riešenia do už existujúcej dokumentácie zjednoduší každodenné činnosti sieťových administrátorov a zároveň poskytne robustnejšie a chybám odolnejšie riešenie.

5.1. Formát zápisu

Pre prehľadnosť a jednoduchosť zápisu konfigurácie bol zvolený formát YAML. Dôvodom zvolenia práve formátu YAML je jeho rozmanitosť, ľahká čitateľnosť a podpora naprieč programovacími jazykmi. YAML ponúka hierarchické štruktúry údajov, tie sú vhodné práve pre konfiguračné súbory, ktoré často potrebujú špecifikovať zložité usporiadanie komponentov. Aj vďaka týmto vlastnostiam je zápis konfigurácie vo formáte YAML v dnešnej dobe možné považovať za svetový štandard [50].

```yaml
Devices:
  - Device: Cisco_Nexus1
    device_os: nxos
    IP_Address: 192.168.1.128
    Interfaces: interfaces_Nexus1
    ACLs:
      - Allow_traffic_A
      - Allow_traffic_C
  - Device: Cisco_IOS1
    device_os: ios
    IP_Address: 192.168.218.129
    Interfaces: interfaces_IOS1
    ACLs:

```

Obrázok 5-1: Názorná ukážka súboru Devices.yaml

Prehľadný a intuitívny zápis vo formáte YAML je dôležitý pre zvýšenie efektivity práce sieťových administrátorov. Hlavné zjednodušenie prichádza v momente, keď pri

modifikácii alebo pridaní pravidla v ACL, sa nemusí administrátor pripojiť pomocou konzole priamo na zariadenie a zadať celý konfiguračný príkaz. Namiesto toho môže využívať vizuálne štruktúrovaný a logicky organizovaný formát YAML, ktorý umožňuje rýchlejšie a presnejšie definovanie pravidiel bez nutného priameho prístupu k zariadeniu. Zápis ACL pravidiel do formátu YAML tiež umožňuje lepšiu správu verzii a auditovateľnosť zmien. Toto je kľúčové pre zabezpečenie integrity a bezpečnosti sieťových nastavení v dynamických prostrediach. Okrem toho, formát YAML umožňuje sieťovým administrátorom používať šablóny a reprodukovateľné komponenty. Administrátori vďaka tomu môžu definovať šablóny pre bežné konfiguračné úlohy a jednoducho tak upraviť konkrétne parametre pre špecifické inštancie.

```yaml
Device: Cisco_Nexus1
Interfaces:
- FastEthernet0/1:
    ACL: Allow_traffic_C
- FastEthernet0/2:
    ACL: Allow_traffic_A

```

Obrázok 5-2: Ukážka aplikácie ACL na rozhrania

5.2. Zdroj pravdy

Koncept zdroju pravdy je v oblasti spravovania rozsiahlych sietí viac ako potrebný. Táto forma dokumentácie predstavuje centralizovanú databázu, ktorá obsahuje všetky aktuálne informácie o konfigurácii siete, jednotlivých zariadeniach a ich jednotlivých rozhraniach[51]. V živej prevádzke, zdroj pravdy slúži ako zdroj informácií, ktorý umožňuje rýchlejšie implementovať zmeny, udržiavať sieť v optimálnej prevádzke a zároveň efektívnejšie reagovať na problémy. Ako už bolo zmienené v predošlých kapitolách, v dynamických prostrediach, v ktorých sa sieťové konfigurácie neustále menia je náročné udržanie konzistencie naprieč rôznymi údajmi. Pri rýchlom raste infraštruktúry, keď sa sieť stáva komplexnejšou a viac sofistikovanou je dôležité zachovať údaje o konfigurácií čo najviac prehľadné. V prípade veľkých dátových centier jednotlivé ACL môžu obsahovať aj viac ako sto pravidiel a z toho dôvodu je extrémne dôležité mať spoľahlivý zdroj pravdy. Pre lepšiu ilustráciu problému, je možné sa zamerať na praktickú konfiguráciu ACL v dátovom centre EmbedIT. Konkrétne ACL na

aplikovaný na vstupe do LOW-SERV zóny obsahuje viac ako sto pravidiel určených na reguláciu prevádzky. Riadenie takéhoto počtu pravidiel bez systematického prístupu môže rýchlo viesť ku chybám. Napríklad, zmeny vykonané na jednom ACL bez aktualizácie ostatných relevantných ACL môžu vytvoriť bezpečnostné riziko alebo neúmyselne blokovať legitímny sieťový traffic. Implementácia zdroju pravdy zároveň minimalizuje riziko chýb tým, že dokáže eliminovať neaktuálne údaje ktoré môžu viesť ku konfiguračným chybám a tak eventuálne spôsobiť nefunkčnosť služby. Okrem už zmienených výhod, aktualizovaný a udržiavaný zdroj pravdy prináša výhody aj v oblasti bezpečnosti[52]. Sieťoví administrátori môžu lepšie identifikovať slabé miesta a rýchlejšie reagovať na potenciálne hrozby. Taktiež je relevantné zmieniť, že celistvý zdroj pravdy zjednodušuje auditovanie siete a zabezpečuje, že všetky vykonané zmeny sú zdokumentované a vyhovujú interným a externým reguláciám.

Ďalším zjednodušením je aj proces hľadania konkrétneho pravidla, ktorý je vo veľkých sieťach často časovo neefektívny, nakoľko administrátor sa musí pripojiť na zariadenia a z veľkého množstva pravidiel vybrať to správne. Preto zavedenie zdroja pravdy pre ACL znamená, že všetky pravidlá a ich zmeny sú centralizovane spravované a dokumentované. Výhodou zápisu je aj zgrupovanie komunikácie z rovnakej zdrojovej IP adresy na cieľovú IP adresu. To znamená, že ak sa v konfigurácií nachádza viacero pravidiel pre rovnaké IP adresy, je možné ich zhluknúť v jednom zápise, ako je možné vidieť na obrázku 5-3. V danom ACL sa nachádza pravidlo *Rule 1*, ktoré obsahuje porty 20, 30 a rozsah portov od 60 do 65. Pre takúto konfiguráciu priamo na zariadení je v ACL potrebné definovať každé použité pravidlo zvlášť pre konkrétne porty. Zápis konfigurácie v použitom formáte umožňuje zápis viacerých portov v jednom pravidle, čo výrazne skráti dĺžku konfiguračného súboru a tak zabezpečí jednoduchý a prehľadný zápis.

Z praxi je známe, že pri veľkom počte pravidiel je náročné udržať prehľad o tom, ktoré pravidlo sa používa na aký účel. A zároveň na zariadeniach je možné vložiť iba poznámku popisujúcu celý ACL a nie konkrétne pravidlá. Pre zvýšenie prehľadu a prístupnosti pre administrátorov bolo zvolené riešenie ktoré obsahuje sekciu *name,* umožňujúcu vložiť názov alebo popis daného pravidla. Toto riešenie uľahčí kolaboráciu, najmä v prípade ak je zariadenie spravované viacerými administrátormi a tak zabezpečí, že každý administrátor bude mať jasný prehľad o konkrétnych pravidlách.

Konkrétny typ formátu zdroja pravdy bol zvolený s ohľadom na jednoduchosť a prehľadnosť pre správu veľkého počtu pravidiel v ACL. Dôraz bol kladený na to, aby bol formát intuitívny a ľahko integrovateľný do existujúcich systémov správy a dokumentácie v spoločnosti EmbedIT. Cieľom bolo dosiahnuť konceptuálnu súladnosť medzi formátom zdroja pravdy a štandardmi, ktoré už boli zavedené a bežne používané v rámci firmy. Z toho dôvodu bol zdroj pravdy navrhnutý presne tak, aby bol čo najviac kompatibilný s ostatnými zdrojmi pravdy používanými v spoločnosti. Touto integráciou sa zároveň aj minimalizuje potreba zásadných zmien v pracovných postupoch administrátorov. Tým sa zabezpečí, že nový zdroj pravdy bude v súlade s prevádzkovanými dokumentačnými konvenciami. Preto sa predpokladá jednoduchá adaptácia zmeny pre sieťových administrátorov. Integrácia tohto zdroja pravdy do systémov automatickej správy ACL sľubuje rýchle a efektívne implementácie bezpečnostných politík. Celková filozofia dizajnu zdroja pravdy bola zameraná na užívateľskú prívetivosť a na minimalizáciu dopadu na bežnú prevádzku. Vďaka intuitívnosti je jeho integrácia do zaužívaných systémov možná bez nutnosti rozsiahleho školenia alebo zložitých prechodných období. Tento prístup nielen zjednodušuje správu ACL pravidiel, ale zároveň poskytuje robustnú platformu pre budúce rozšírenia a aktualizácie.

```yaml
ACL:
- ACL_name: Allow_traffic_A
  remark: ACL allowing Data Traffic - server A
  prefix: 192.168.1.0/24
  type: extended
  rules:
      - name: Rule 1 - server A
        grant: permit
        source_addresses: 192.168.1.10/32
        source_ports:
        protocols: tcp
        destination_addresses: 10.10.10.10/32
        destination_ports: 20,30,60-65
      - name: Rule 2 - server B
        grant: permit
        source_addresses: 10.10.10.128/32
        source_ports:
        protocols: tcp
        destination_addresses: 192.168.1.10/32
        destination_ports: 50,10-30
      - name: Allow ICMP-NOGENERATE
        grant: permit
        source_addresses: any
        source_ports:
        protocols: icmp
        destination_addresses: any
        destination_ports:
      - name: Deny ALL-NOGENERATE
        grant: deny
        source_addresses: any
        source_ports:
        protocols: tcp
        destination_addresses: any
        destination_ports:

```

Obrázok 5-3: Ukážka definície ACL

5.3. Hierarchická štruktúra

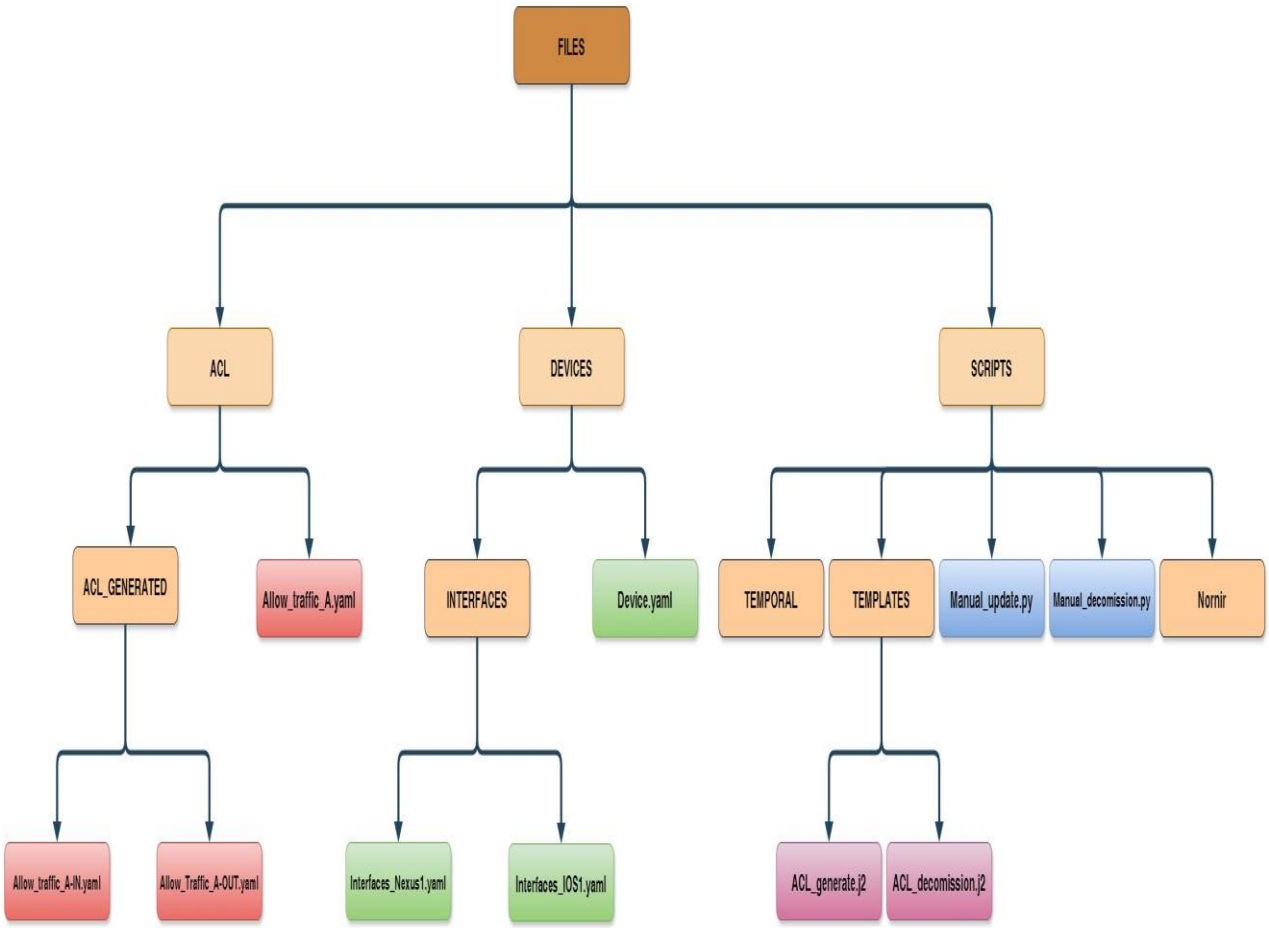
S ohľadom na aktuálnu štruktúru používanú v dokumentácií spoločnosti EmbedIT, ako najvhodnejšie riešenie prišlo riešenie s hierarchickou štruktúrou rozdelenou do troch hlavných častí ako je možné vidieť na obrázku 5-4. Prvá zložka časť nesie názov *ACL* a pozostáva z viacerých konfigurácií jednotlivých ACL. V ukážke je použitý ACL s názvom *Allow_Traffic_A*, skladajúci sa z viacero pravidiel a je definovaný v súbore YAML. V aktuálnej zložke ACL sa nachádza aj zložka s názvom *ALL_GENERATED*

a v nej sa nachádzajú už vygenerované ACL listy v smere IN a OUT pre každý ACL list pomocou skriptu *ACL-GEN.py*. Tento proces bude bližšie popísaný v nasledujúcej kapitole. Zatiaľ je iba potrebné porozumieť, že vygeneroval konfiguráciu ACL v smere IN a OUT, ktoré budú následne aplikované na konkrétne rozhranie. Táto štruktúra udržuje prehľad a poriadok v dokumentácií a umožňuje administrátorom intuitívne modifikovať iba YAML súbory.

Druhá zložka s názvom *DEVICES* obsahuje zdrojový YAML súbor s názvom Devices.yaml v ktorom sú špecifikované jednotlivé zariadenia použité v topológii. YAML súbor je veľmi intuitívny a ako je možné vidieť v obrázku 5-1, obsahuje iba názov zariadenia, operačný systém zariadia, IP adresu, odkaz na súbor s rozhraniami a aplikované ACL. Špecifikácia operačného systému je kľúčová pre potreby zvolenia konkrétnej šablóny v konfigurácií. To je potrebné z dôvodu, že v topológii sa nachádzajú zariadenia s rôznymi operačnými systémami, čo implikuje potrebu použitia rozdielnej syntaxi. IP adresa je v konfigurácii potrebná hlavne pre potreby knižnice Nornir, ktorá zabezpečuje vzdialené pripojenie pomocou SSH na zariadenie. Zložka *DEVICES* obsahuje mimo zdrojového súboru *Devices.yaml* aj zložku s názvom *INTERFACES*. Ako už názov napovedá, jedná sa o zložku s rozhraniami zariadení. Pre zachovanie prehľadnosti a jednoduchosti, každé zariadenie disponuje vlastným súborom v ktorom sú špecifikované konkrétne konfigurácie. Momentálne súbor podporuje iba špecifikáciu ACL pravidiel, ale pre budúce potreby je možné jednoducho priradiť k mapovaniu aj rôzne vlastnosti, ako IP adresa, Routing protokoly, VLAN, Quality of Service a podobne, ktoré budú rovnakým procesom nakonfigurované na zariadenie.

Tretia zložka s názvom *SCRIPTS* obsahuje všetky potrebné Python skripty na vytvorenie ACL pravidiel a zároveň aplikovanie hromadnej konfigurácie. Tieto skripty sú nastavené tak, nech sieťoví administrátori nemajú potrebu v nich vykonávať nijaké zmeny. Zložka *SCRIPTS* obsahuje nie len konfiguračné skripty ale aj ďalšie zložky s názvami *TEMPLATES, TEMPORAL a NORNIR*. Ako už prvý názov napovedá, jedná sa o zložku obsahujúcu šablóny. Pre potreby tejto práce bol použitý šablónovací jazyk Jinja2. Druhá zložka s názvom *TEMPORAL* obsahuje dočasné súbory vytvorené počas aplikačnej pipeline. Táto zložka slúži predovšetkým na účely hľadania chýb medzi jednotlivými procesmi alebo ako zdroj informácií pre prípad aplikácie pokročilejších skriptov. Predpokladá sa, že s obsahom tejto zložky sa nebude manipulovať, keďže slúži

iba ako dočasná pamäť pre jednotlivé procesy v pipeline. Pri modifikácií niektorého zo súborov by mohla nastať zmena medzi zdrojom pravdy a reálnou konfiguráciou na zariadeniach. Nesúlad medzi nimi môže spôsobiť vážne nezhody, ktoré môžu eventuálne viesť k narušení bezpečnosti a chodu prevádzky. Posledná zložka Nornir obsahuje konfiguračné súbory pre správne fungovanie Norniru, ako *config.yaml* alebo *hosts.yaml*.



Obrázok 5-4: Hierarchická štruktúra

6. IMPLEMENTÁCIA PROCESU

Hlavná časť pozostáva z návrhu a implementácie komplexného riešenia s cieľom automatizovať procesy pre vytváranie, modifikovanie a odstraňovanie ACL a ich nasledovné aplikovanie na sieťové zariadenia. Implementácia ACL pravidiel z centralizovaného zdroja pravdy je proces, ktorý začína extrakciou potrebných údajov z tohto zdroja. Tieto informácie sú následne spracované skriptom v jazyku Python, ktorý využíva knižnicu Nornir na distribúciu a aplikáciu konfiguračných zmien na cieľové sieťové zariadenia. V nasledujúcich podkapitolách je podrobne popísaný výber násstrojov, ktoré boli použité pri návrhu riešenia ako aj detailný popis implementácie navrhnutého systému a jeho otestovanie v simulačnom prostredí.

6.1. Výber vhodného nástroja

Výber vhodného automatizačného nástroja bol vykonaný s ohľadom na aktuálne technické možnosti sieťových prvkov, ako aj s ohľadom na jednoduchosť použitia a existujúce automatizačné procesy implementované v spoločnosti EmbedIT. Prvým a zásadným limitujúcim faktorom, ktorý ovplyvnil rozhodovací proces, bola nekompatibilita sieťových zariadení s modernými API rozhraniami. Zariadenia použité v dátových centrách sú staršieho typu a neumožňujú priame API pripojenia, ktoré by umožnili sofistikovanejšie formy automatizácie. Tieto rozhrania umožňujú rozsiahlejšiu automatizáciu a správu, čím signifikantne zvyšujú efektivitu správy sieťových zariadení, ako napríklad Cisco ACI [53]. Preto bola jedinou dostupnou možnosťou automatizácia cez konzolové pripojenie používajúce protokol SSH. Táto situácia predstavuje významné obmedzenie, pretože aj keď SSH poskytuje solídnu platformu pre vykonávanie príkazov na diaľku, nie je tak flexibilná ako novšie metódy, ktoré využívajú API alebo podobné technológie. Ďalším faktorom boli aj dlhodobo zavedené konvencie v dokumentácií a procesoch a s tým spojené aj skúsenosti sieťových administrátorov v tíme.

Po analýze aktuálneho stavu zariadení v dátovom centre a porovnaní dostupných možností bol zvolený postup využívajúci programovací jazyk Python spolu s knižnicou Nornir. Ako bolo už spomenuté v predchádzajúcej kapitole, Python umožňuje vytvárať skripty, ktoré môžu dynamicky interagovať so sieťovými zariadeniami. Nornir zase poskytuje robustné riešenie pre vykonávanie týchto skriptov na rôznych zariadeniach a rôznych operačných systémoch. Okrem toho, zvolenie práve kombinácie Pythonu

a Norniru je výhodné v tom, že sieťový administrátori, ktorí s ním budú pracovať, už majú predošlé skúsenosti a tak môžu využívať už známe nástroje a techniky. To znamená, že integráciou tejto metódy sa zároveň aj minimalizuje čas potrebný na naučenie a adaptáciu tejto technológie. Hlavnou výhodou tohto riešenia je, že Python umožňuje aj následnú modifikáciu skriptov pre rôzne účely a tak môže byť hocikedy upravený podľa preferencií a potrieb administrátorov. Na záver je teda dôležité poznamenať, že rôznorodosť Pythonu otvára dvere pre ďalšie možnosti rozšírenia a inovácie v budúcnosti.

6.2. Generovanie ACL

Konfiguračný problém s ktorým sa sieťový administrátori stretávajú veľmi často je nastavenie potrebných ACL pravidiel v oboch smeroch. Tento krok robí z jednoduchej úlohy zložitú v tom zmysle, že ak sieťový administrátor povolí traffic jedným smerom, je potrebné povoliť aj druhým smerom. Pre uvedenie príkladu, ak chce administrátor povoliť traffic z externej siete, musí špecifikovať IP adresu a port zdrojovej a cieľovej adresy v smere do internej siete. To isté je potrebné vykonať aj pre vracajúci sa traffic v smere do externej siete. Tento proces je nielenže časovo neefektívny, neprehľadný ale aj veľmi náchylný na chyby.

Pre uľahčenie tohto komplexného procesu bol vytvorený a otestovaný skript, v ktorom administrátor iba jednoducho špecifikuje v ACL pravidlo a konfigurácia pre smer IN (do internej siete) a OUT (do externej siete) bude automaticky vytvorená na základe adresného rozsahu internej siete. Logika tohto procesu je jednoduchá, administrátor iba špecifikuje názov ACL, remark (popis) a prefix (adresný rozsah) pre celý ACL a potom následne názov pravidla, typ ACL, protokol, zdrojovú adresu, cieľovú adresu a cieľový port pre jednotlivé pravidlá v ACL. Je dôležité podotknúť, že v tejto konfigurácií nie je potrebné zadávať zdrojový port, nakoľko nie je potrebný. Skript automaticky vyhodnotí, ktorá adresa sa nachádza v adresnom rozsahu siete a na základe toho správne vyhodnotí ktoré pravidlá budú v smere IN a ktoré v smere OUT. Skript nielenže vyhodnotí smer v ktorom má byť dané pravidlo aplikované ale aj vytvorí pravidlo v opačnom smere, ktoré povoľuje vracajúci sa traffic. Výsledkom tohto procesu sú 2 súbory YAML, jeden obsahujúci ACL ktorý bude aplikovaný v smere IN a druhý ktorý bude aplikovaný v smere OUT.

Pre tento prípad je možné použiť dve varianty. Prvá alternatíva je vhodná pre prísnejšie kontrolovanú prevádzku, kedy skript vytvorí opačné pravidlo špecificky určené pre vracajúci sa traffic z rovnakej cieľovej IP adresy na rovnakú zdrojovú IP adresu. Toto je vhodné napríklad ak sa jedná o prevádzku mimo internú sieť dátového centra alebo ak je dôležité aby pravidlo povoľujúce vracajúci sa traffic zbieralo hodnoty o tom koľko krát toto pravidlo zabránilo prechodu určitej prevádzky. Druhá alternatíva prináša zjednodušenie, v ktorom je návratový traffic povolený iba v jednom pravidle. Toto je zaistené pomocou takzvaných ephemeral portov, ktoré je možné nazvať aj ako dočasné porty[54, 55]. Tieto porty sú dynamicky pridelené na strane klienta a sú používané počas trvania jednorazovej komunikačnej session. V kontexte návratového trafficu, kedy server odpovedá klientovi, ephemeral port slúži ako destinácia pre prichádzajúce pakety. Po úspešnom nadviazaní spojenia a priradení ephemeral portu na klientovej strane, server použije tento port ako cieľovú adresu pre odosielanie dát späť klientovi. Tento proces zabezpečuje, že komunikácia medzi klientom a serverom je efektívne spravovaná a že všetky odpovede sú presne smerované na správne session. To je dôležité v prostrediach, v ktorých je počet dostupných portov limitovaný a tým pádom je ich efektívne využívanie kritické pre plynulý chod dynamickej prevádzky. Napríklad v aplikáciách využívajúcich vysokú úroveň paralelných požiadaviek, ephemeral porty eliminujú konflikty a umožňujú rýchlejšie spracovanie viacerých požiadaviek súčasne. Ďalšou výhodou použitia ephemeral portov je zvýšená bezpečnosť. Keďže porty sú pridelené náhodne a existujú len počas trvania sieťovej session, je zložitejšie pre útočníkov predpovedať, ktoré porty budú použité, a tým pádom zneužiť aktívne sieťové spojenia. Typicky sú pridelené z určitého rozsahu čísel, ktorý je špecifický pre operačný systém a po ukončení komunikácie sú opäť uvoľnené. Rozsah ephemeral portov je v systémoch Windows od 49152 do 65535 a v UNIXových systémoch už od 32768[56]. Nakoľko podstatná časť IT infraštruktúry v dátovom centre využíva práve UNIXový systém Linux, bol zvolený rozsah už od čísla 32768.

Výhodou implementácie tohto riešenia je aj ich rozsiahle použitie v cloudových prostrediach v podobe Network Access Control Listov. Toto použitie je doporučené aj v oficiálnej dokumentácii AWS[57]. Nakoľko spoločnosť EmbedIT aktívne využíva AWS a v blízkej budúcnosti plánuje migráciu do cloudu, táto vymoženosť signifikantne uľahčí proces migrácie.

Výsledkom tohto procesu sú 2 súbory YAML, jeden obsahujúci ACL ktorý bude aplikovaný v smere IN a druhý ktorý bude aplikovaný v smere OUT.

Je kľúčové podotknúť, že v reálnej prevádzke sa uplatňujú aj pravidlá, ktoré nepodliehajú nutnosti mať rozdielnu konfiguráciu v oboch smeroch. Tým je často vyskytujúce sa pravidlo povoľujúce traffic pomocou protokolu ICMP. Toto pravidlo zvykne byť často povolené hlavne z dôvodu monitoringu zariadení, na ktoré sa v periodických intervaloch odošle ping na zariadenie. Tým sa môže napríklad zisťovať jeho dostupnosť alebo merať latencia, ktorá sa zvyčajne zhromažďuje do grafov a používa sa na vyhodnocovanie kvality siete alebo či v sieti nedošlo k nejakej anomálií. V sieti spoločnosti EmbedIT je zvykom, že toto pravidlo je povolené zo všetkých externých sietí dátového centra do internej siete na rozhraní. Ďalším príkladom, môže byť pravidlo ktoré zakazuje prevádzku vo všetkých smeroch. Toto pravidlo sa zvyčajne používa ako posledné a pragmaticky znamená, že všetok traffic, ktorý nie je povolený v skoršie definovaných pravidlách, bude explicitne zakázaný. Môže sa zdať, že toto pravidlo je nadbytočné, pretože ak sa v ACL nachádza definované pravidlo povoľujúce určitú prevádzku, nachádza sa tam zároveň aj implicitné pravidlo, ktoré zakáže všetko ostatné a opačne. V tomto prípade ale nie je možné zbierať hodnoty, koľko krát toto pravidlo zabránilo prechodu určitej prevádzky, ktoré sú pre sieťových architektov potrebné pre udržiavanie bezpečnosti v sieti. Pre tieto možnosti a ďalšie iné sa nachádza možnosť použiť v názve pravidla slovo *NOGENERATE*, ktoré zabezpečí, že k pravidlu nebude vygenerované zrkadlové pravidlo ale ponechá sa pre oba smery tak ako ho zadal administrátor. Spustenie skriptu je pomocou špecifikácie daného ACL v príkazovom riadku *Usage: python ACL-gen.py <ACL_Name>* alebo pre prípad striktnejšej varianty *Usage: python ACL-gen-strict.py <ACL_Name>.*

```
30    - name: general_inbound_tcp
31      grant: permit
32      source_addresses: any
33      source_ports:
34      protocols: tcp
35      destination_addresses: any
36      destination_ports: 32768-65535
37      direction: in
```

Obrázok 6-1: Všeobecné pravidlo povoľujúce návratový traffic

Podľa dohody sieťovými architektmi a na základe ich skúseností som zvolil riešenie manipulácie s ACL, ktoré sa skladá z troch častí:

1. Celková synchronizácia ACL so zariadením
2. Manuálna aktualizácia ACL na zariadenie
3. Manuálne odstránenie ACL zo zariadenia

6.3. Celková synchronizácia so zariadením

Tento postup reprezentuje hromadnú synchronizáciu všetkých definovaných a aplikovaných ACL v zdroji pravdy na konkrétne zariadenie. Tento proces môže sieťový administrátor vykonať manuálne spustením skriptu alebo je možné nastaviť automatické vykonanie tohto procesu v prednastavenom čase bez potreby manuálnej interakcie. Na toto môže byť použitý napríklad softvérový nástroj *Cron*, ktorý sa používa v operačných systémoch typu UNIX na plánovanie periodicky sa opakujúcich úloh[58]. Toto zabezpečí najnovšiu konfiguráciu nastavenú na zariadeniach.

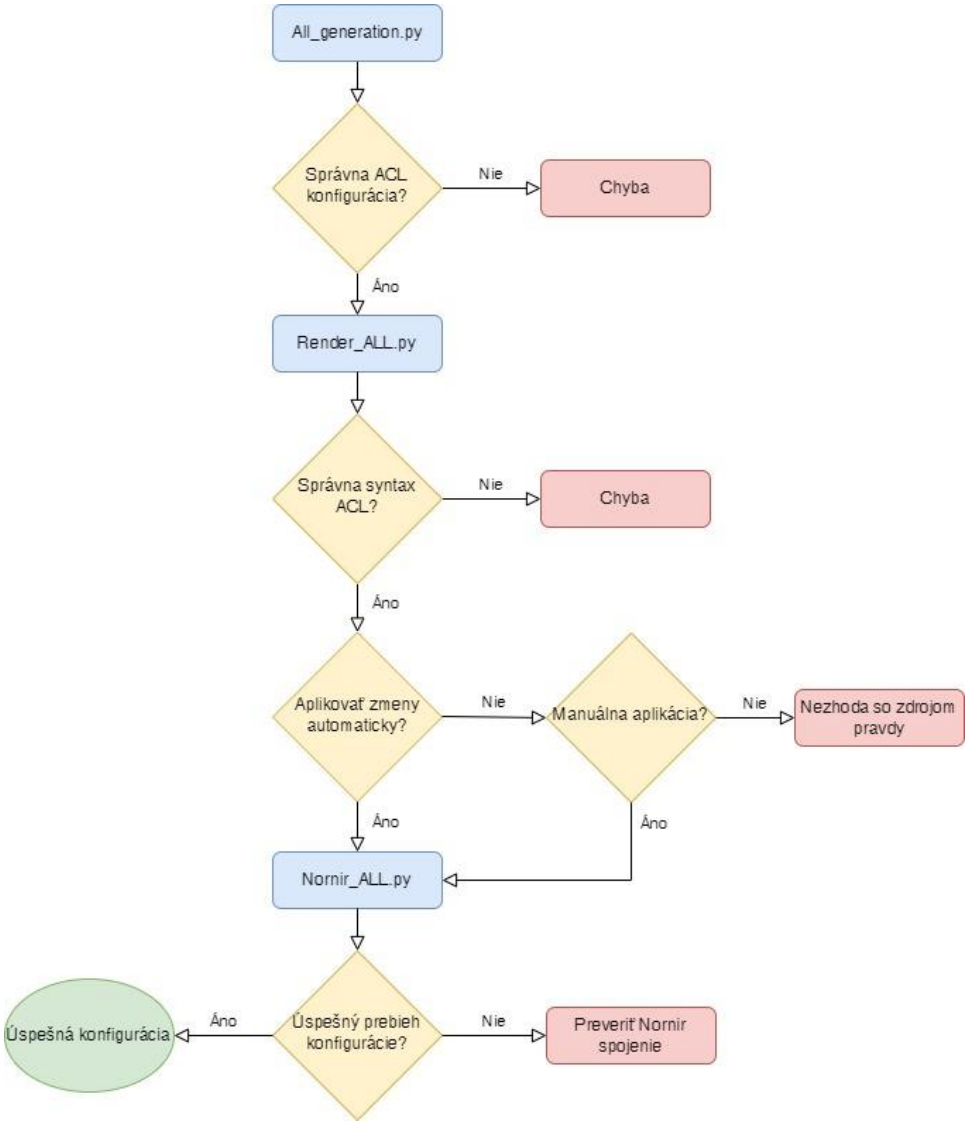
Spustenie skriptu sa vykonáva z terminálu a je veľmi intuitívne, administrátor iba špecifikuje názov zariadenia ktoré má byť aktualizované, *`Usage: All_generation.py <Device_Name>`* Od momentu ako administrátor zadá tento príkaz, už automat vykonáva všetky potrebné úkony. Skript v úvode načíta konfiguračný súbor YAML obsahujúci údaje o použitých sieťových zariadeniach v architektúre. Ďalej si vyhľadá konkrétne zariadenie zadané administrátorom, z ktorého extrahuje všetky potrebné hodnoty o definovaných ACL na zariadení a o súbore v ktorom sa nachádzajú detaily o rozhraniach. V ďalšom kroku načíta príslušný súbor o rozhraniach na zariadení a skontroluje prítomnosť ACL. Súbežne s tým overí aj ich zhody so záznamami v hlavnom konfiguračnom súbore so zariadeniami. Ak sú všetky aplikované ACL na rozhraniach prítomné, extrahuje hodnoty z rozhraní a ako výstup vytvorí konsolidovaný výstup obsahujúci informácie o zariadení, jeho rozhraniach a použitých ACL. Jedná sa o dočasný súbor obsahujúci túto konfiguráciu vo formáte YAML. Automaticky po dokončení sa spustí ďalší skript pomocou príkazu `subprocess`. Tento skript ako vstup vezme dočasný konfiguračný súbor a šablónu Jinja2, ktoré použije na vygenerovanie konfigurácie. Tento skript zároveň overí syntaktickú správnosť ACL konfigurácií, vrátane použitých IP adries, protokolov, portov, typu a smeru, aby sa predišlo

konfiguračným chybám. V prípade, ak by boli chyby lokalizované, proces sa zastaví a upozorní v ktorom pravidle sa vyskytla konkrétna chyba. Ak je konfigurácia bezchybná, skript pokračuje s renderovaním šablóny. Skript takisto obsahuje aj funkciu zabezpečujúcu konverziu IP adries vo formáte CIDR na IP adresy s wildcard maskami. Tento proces je dôležitý, nakoľko použité zariadenia neakceptujú zadanie IP adries v CIDR formáte. Vygenerovaný výstup je rozdelený na 2 časti, prvá časť spresňuje použité zariadenie a druhá časť už obsahuje konkrétnu konfiguráciu, ktorá už je v syntakticky jednotnej podobe pre aplikáciu na daný typ zariadenia. Tento výstup je nielen uložený ako dočasný súbor, ktorý bude použitý zdroj pre Nornir skript aplikujúci konfiguráciu na zariadenie, ale aj ako viditeľný výstup do terminálu pre kontrolu administrátorom. Toto slúži na vytvorenie exekučného plánu, ktorý predstavuje súhrn zmien, ktoré budú vykonané na základe dočasného konfiguračného súboru. Tento krok ponúka možnosť administrátorovi preskúmať zmeny skôr ako sa aplikujú na zariadení. Po preskúmaní sa ďalej v tomto kroku ponúka možnosť, či chce danú konfiguráciu automaticky aplikovať na zariadenie alebo nie. Spomínaný prístup v procese kontroly a následnej voliteľnej aplikácie je inšpirovaný s pracovným postupom, ktorý sa využíva v nástroji Terraform, konkrétne s príkazmi `terraform plan` a `terraform apply`. Tento postup bol zvolený ako súčasť procesu nakoľko je veľmi obľúbený v oblasti spravovania infraštruktúry a preto.

V prípade ak sa administrátor rozhodne nepokračovať s automatickou aplikáciou, ponúkajú sa mu dve možnosti. Buď môže manuálne spustiť skript, ktorý pomocou Norniru aplikuje konfiguráciu, alebo konfiguráciu nahrať manuálne priamo na zariadenie. Pri zvolení tohto postupu ale vzniká riziko ľudskej chyby a to môže mať za následok, že konfigurácia zariadenia sa nemusí zhodovať so zdrojom pravdy. Táto situácia predstavuje obrovské bezpečnostné riziko, preto sa táto možnosť neodporúča. Na tento krok upozorní aj bezpečnostná hláška, ktorá notifikuje administrátora ak sa rozhodne pre túto možnosť.

Diagram na obrázku 6-2 vizualizuje postup automatizácie konfigurácie ACL pomocou sekvencie troch skriptov. V diagrame je ilustrované ako každý skript kontroluje správnosť konfigurácie, v prípade výskytu chyby je celý proces zastavený v danom kroku. Diagram síce prezentuje iba proces automatickej synchronizácie, ale analogicky

tomu odpovedajú aj procesy manuálnej aktualizácie a odstránenia, v ktorých je postup identický.



Obrázok 6-2: Diagram procesu aplikácie ACL

Je dôležité podotknúť, že v bežnej praxi je výskyt manuálnych chýb veľmi častý a niekedy je náročné ich lokalizovať aj pri dôkladnej kontrole. Vhodný príkladom môže byť napríklad nesprávny formát IP adresy alebo nesprávne zapísané slovo *permit*. Toto by spôsobilo, že dané pravidlo by správne aplikované na zariadení a chyba by sa zistila až pri samotnej implementácii na zariadenie. To je samozrejme už neskoro a mohlo by to mať za následok výpadok služieb až do momentu, pokiaľ by táto chyba nebola odhalená

a napravená administrátorom. Aby sa predišlo tomuto riziku konfiguračnej chyby, každý skript obsahuje proaktívne riešenie v podobe validátora, ktorý vyhodnotí či zadané údaje sú v správnom formáte. Nadväzujúc na vyššie spomenutý príklad s IP adresou, validátor by odhalil nesprávny formát IP adresy, prípadne výskyt nežiaduceho znaku, upozorní na chybu a zastaví celý proces. Táto vymoženosť proaktívne zabraňuje výskytu konfiguračnej chyby z nepozornosti a tak výrazne zvyšuje odolnosť a robustnosť daného riešenia.

6.4. Manuálna aktualizácia ACL na zariadenie

Druhá možnosť predstavuje aktualizáciu iba jedného ACL, bez nutnosti manipulovať s inými ACL. Tento variant je vhodný v prípade, ak administrátor vytvoril nový ACL alebo vykonal zmeny v pravidlách v už existujúcom ACL, v ktorom napríklad pridal alebo zmenil určité pravidlo. V tomto prípade nie je absolútne potrebné vykonávať synchronizáciu všetkých ACL. Po dokončení úprav v zdroji pravdy, administrátor spustí skript *Usage: python Manual_update.py <ACL_Name>*. Pre zavolanie skriptu je nutné špecifikovať iba názov ACL, nakoľko názvy ACL sú unikátne v celej infraštruktúre, skript iteruje cez konfiguračný súbor zariadení a automaticky vyhodnotí na ktorom zariadení sa nachádza. To zjednodušuje praktické použitie pre administrátora. Od tohto kroku, je už postupovanie automatu analogické s celkovou synchronizáciou, nasleduje sekvencia skriptov ukončená možnosťou kontroly a automatickej aplikácie pomocou Norniru po vygenerovaní súhrnu zmien, ktoré budú vykonané.

6.5. Manuálne odstránenie ACL zo zariadenia

Tretí variant sa používa pre možnosť ak je potrebné ACL odstrániť zo zariadenia bez potreby manipulácie s ostatnými ACL. Jeho použitie je taktiež veľmi intuitívne. Administrátor spustí skript *Usage: python Manual_decomission.py <ACL_Name>*, kde do príkazového riadku definuje iba ACL ktorý má byť zo zariadenia odstránený. Pre zachovanie synchronizácie medzi zdrojom pravdy a zariadením, sa pri spustení tohto skriptu ACL odstráni súčasne zo zariadenia a zo zdroju pravdy. Konkrétne sa odstráni zo zariadenia, v ktorom je definovaný a z rozhrania, na ktorom je aplikovaný. Proces exekúcie skriptu sa začína keď je konkrétny ACL špecifikovaný v príkazovom riadku. Skript prehľadá všetky zariadenia a rozhrania v konfiguračných súboroch a ak sa ACL nájde, odstráni ho z oboch súborov. Toto zachová súlad medzi zdrojom pravdy

a zariadením. Pre možnosti reaplikácie alebo možnej úpravy v budúcnosti, ACL stále ostane vytvorený v zložke ACL. Proces exekúcie je taktiež analogický k predchádzajúcim skriptom.

```
Generated Configuration:

ip access-list extended Allow_traffic_A-AUTOMAT-inbound
remark ACL allowing Data Traffic - server A for inbound traffic
1 permit tcp 192.168.1.10/32 10.10.10.10/32 eq 20
2 permit tcp 192.168.1.10/32 10.10.10.10/32 eq 30
3 permit tcp 192.168.1.10/32 10.10.10.10/32 range 60 65
4 permit icmp any any
5 permit tcp any any
6 permit tcp any any range 32768 65535
7 permit udp any any range 32768 65535
ip access-list extended Allow_traffic_A-AUTOMAT-outbound
remark ACL allowing Data Traffic - server A for outbound traffic
1 permit tcp 10.10.10.128/32 192.168.1.10/32 eq 50
2 permit tcp 10.10.10.128/32 192.168.1.10/32 range 10 30
3 permit icmp any any
4 permit tcp any any
5 permit tcp any any range 32768 65535
6 permit udp any any range 32768 65535
interface FastEthernet0/2
ip access-group Allow_traffic_A-AUTOMAT-inbound in
ip access-group Allow_traffic_A-AUTOMAT-outbound out
Do you want to apply this configuration to the device? (YES/NO):
```

Obrázok 6-3: Výpis vygenerovanej konfigurácie

6.6. Šablóna JINJA2

Šablóny Jinja2 sú navrhnuté tak, aby boli ľahko pochopiteľné a veľmi dobre čitateľné. Použitie Jinja2 v hromadnej konfigurácií siete prispieva nielen k robustnému riešeniu, ktoré je odolné voči chybám ale aj v efektívnej manipulácii s jeho obsahom[59]. Jinja2 umožňuje vysokú mieru prispôsobenia, čo otvára dvere pre jednoduchú modifikáciu súboru v prípade pridania pokročilých funkcií.

V použitých šablónach sa využíva aj jednoduchá programovacia logika, vďaka ktorej sa zaistí použitie správnej syntaxe pre konkrétne zariadenie. Je dôležité zmieniť, že šablóny sú rozdielne pre skript pridávajúci ACL pravidlá a pre skript, ktorý tieto pravidlá odstraňuje. Toto je hlavne z dôvodu, že v procese odstraňovania nie je potrebné špecifikovať jednotlivé pravidlá, ale je postačujúce iba odstrániť celý ACL a jeho

použitie na rozhraní. Skript zabezpečí prevod dočasného YAML súboru s konfiguráciou na syntakticky korektnú verziu pripravenú na aplikáciu na konkrétne zariadenie. Ako už bolo spomenuté v predošlej kapitole, v infraštruktúre dátových centier spoločnosti EmbedIT sa nachádzajú zariadenia rôznych výrobcov a s rôznymi operačnými systémami. To znamená, že každá konfigurácia pre tieto zariadenia nie je identická ale požaduje určite syntaktické zmeny. Napríklad, infraštruktúra zahŕňa viacero zariadení od spoločnosti Cisco, časť z nich beží na operačnom systéme NXOS a druhá časť na IOS. Rozdiel v syntaxi pre tieto zariadenia nie je síce veľký, ale konfigurácia pre NXOS by sa neaplikovala správne na zariadenie IOS. V inom dátovom centre sa nachádza dokonca topológia využívajúca HPE switch. Z tohto dôvodu šablóna obsahuje logickú štruktúru, v ktorej sa podľa špecifikácie operačného systému v zdroji pravdy zvolí správa šablóna, z ktorej sa následne výsledná konfigurácia vygeneruje. Celkovo teda šablóna obsahuje pre 3 rôzne zariadenia. Po úvodnom vypísaní názvu, operačného systému a IP adresy zariadenia, skript iteruje cez ACL na danom zariadení a vytvára konfiguračné riadky pre každé použité pravidlo. Pravidlá sú ďalej rozdelené podľa zdrojových a cieľových adries, zdrojových a cieľových portov. Každé pravidlo je zároveň sekvenčne očíslované presne podľa poradia v ako sú zadané v konfiguračnom súbore. V druhej časti skript iteruje cez jednotlivé rozhrania zariadenia a priradí k nim príslušné ACL v konkrétnom smere, či už v smere inbound alebo outbound. Na záver táto výsledná konfigurácia obsahuje všetky potrebné príkazy na aplikáciu ACL na dané sieťové zariadenie.

```
Device : {{ device_1 ['device_name'] }}
Device Operation system : {{ device_1 ['device_os'] }}
{% if 'nxos' == device_1 ['device_os'] -%}
        {# ------------------------ #}
        {# Config for NX-OS  #}
        {# ------------------------ #}
IP_Address : {{ device_1 ['IP_Address'] }}
!
{% set counter = [0] %}
{% for acl in device_1['ACLs'] %}
ip access-list extended {{ acl['ACL_name'] }}
remark {{ acl['remark'] }}
{% for rule in acl['rules'] %}
  {% set source_ports = rule['source_ports'].split(',') if rule['source_ports'] is string and rule['source_ports'] is not none
     else [rule['source_ports']] %}
  {% set destination_ports = rule['destination_ports'].split(',') if rule['destination_ports'] is string
     and rule['destination_ports'] is not none else [rule['destination_ports']] %}
  {% for port in source_ports %}
    {% for dest_port in destination_ports %}
      {{ counter[0] }} permit {{ rule['protocols'] }} {{ rule['source_addresses'] }} {% if port %}
      {% if '-' in port|string %}range {{ port|replace('-', ' ') }}{% else %}eq {{ port }}{% endif %}
      {% endif %}{{ rule['destination_addresses'] }} {% if dest_port %}{% if '-' in dest_port|string %}range {{ dest_port|replace('-', ' ') }}
      {% else %}eq {{ dest_port }}{% endif %}{% endif %}
      {% set counter = [counter[0] + 1] %}
    {% endfor %}
  {% endfor %}
{% endfor %}
{% endfor %}

{% set acl_directions = {} %}
{% for acl in device_1.ACLs %}
    {% set _ = acl_directions.update({ acl.ACL_name: acl.direction }) %}
{% endfor %}
{% for interface_name, interface_details in device_1.Interfaces.items() %}
    {% set acl_infos = interface_details.ACL %}
interface {{ interface_name }}
    {% for acl_name in acl_infos %}
        {% set direction = acl_directions[acl_name] if acl_name in acl_directions else '' %}
ip access-group {{ acl_name }} {{ direction }}
    {% endfor %}
{% endfor %}

```

Obrázok 6-4: Šablóna Jinja2 pre NX-OS

Tento proces taktiež výrazne zjednodušuje postup aplikácie ACL pre sieťových administrátorov, nakoľko odstraňuje potrebu znalosti konkrétnej syntaktickej formy pre zariadenia od rôznych výrobcov. Šablóna ponúka na výber 3 rôzne typy zariadení, čo je pre potreby sieťovej infraštruktúry spoločnosti EmbedIT dostačujúce. Jedná sa o zariadenia Cisco s operačnými systémami NX-OS a IOS a o zariadenia HPE. Zároveň jednoduchá a užívateľsky prívetivá štruktúra šablóny otvára dvere pre jednoduchú modifikáciu a prípadné pridanie novej šablóny pre iný typ zariadenia ak by bolo do infraštruktúry pridané.

```yaml
---
Cisco_Nexus1:
  hostname: 192.168.40.128
  port: 22
  platform: cisco_nxos_ssh
  connection_options:
    netmiko:
      extras:
        device_type: "cisco_nxos_ssh"
Cisco_IOS1:
  hostname: 192.168.40.129
  port: 22
  platform: cisco_ios_ssh
  connection_options:
    netmiko:
      extras:
        device_type: "cisco_ios_ssh"

```

Obrázok 6-5: Definícia hostov pre knižnicu Nornir

6.7. Aplikácia ACL na zariadenia pomocou knižnice

Finálny krok pozostáva z Python skriptu, ktorý využíva knižnicu Nornir a pomocou nej sa s pripája na CLI ku konkrétnemu zariadeniu. Tieto skripty sú rozdelené taktiež do troch častí podľa zámeru administrátora. To je z dôvodu, že každý proces vyžaduje iné úkony a ich exekúcia by jednak mohla byť redundantná alebo dokonca až nežiaduca. Avšak vo všetkých variantoch je logika postupovania rovnaká. V prvom kroku sa vytvorí spojenie so zariadením a začnú sa vykonávať potrebné príkazy. Tento sa proces sa začína načítaním konfiguračného súboru, v ktorom sú definované podrobnosti inventára, parametre pripojenia a nastavenia vykonávania úloh. Inventár je veľmi flexibilný a obsahuje informácie o hostoch a skupinách, čo umožňuje jednoducho filtrovať a zoskupovať rôzne zariadenia. Ako je ilustrované na obrázku 6-5, je definovaný názov zariadenia, IP adresa, port a typ zariadenia. Táto konfigurácia je dostatočná pre testovacie účely, avšak v reálnej praxi je možné zoskupiť zariadenia podľa rôznych parametrov, čo výrazne zjednoduší použitie. Dobrým príkladom môže byť zoskupenie zariadení podľa ich konkrétnej role v dátovom centre. Konkrétne zariadenia sú filtrované pomocou vstavenej schopnosti inventára ako: `nr_filtered = nr.filter(name=device_name)`. Je dôležité poznamenať, že Nornir umožňuje aj

pokročilé filtrovanie pomocou logických operátorov, čo umožní vytvoriť aj komplexnejšie filtračné podmienky.

Pre celkovú synchronizáciu sa v prvom kroku vykoná príkaz *show ip access-lists | include AUTOMAT*, tento príkaz vypíše aktuálne definované ACL na zariadení, ktoré sú spravované automatom. V tomto kroku je veľmi dôležité zmieniť, že určité ACL, za žiadnych okolností nemôžu byť zo zariadenia odstránené. Napríklad ACL, povoľujúci konektivitu Norniru so zariadením. Pri odstránení tohto ACL by mohlo nastať, že by sa v strede procesu prerušila konektivita a tak by zariadenie ostalo v nedokonfigurovanom stave. Táto situácia by mohla spôsobiť seriózne dôsledky na chod služieb. Pre tento dôvod automat manipuluje iba s určitými ACL, ktoré sú na to určené. Zároveň je tak možné ponechať na zariadení ACL, ktoré sú kritické pre správny chod zariadenia.

```
R1#show ip access-lists | include AUTOMAT
Extended IP access list Allow_traffic_C-AUTOMAT
R1#
R1#
R1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#
R1(config)#no ip access-list extended Allow_traffic_C-AUTOMAT
R1(config)#
R1(config)#end
```

Obrázok 6-6: Ukážka automaticky odoslaných príkazov na zariadení

Skript si načíta zobrazené ACL a vykoná s nimi operáciu, ktorá ich postupne odstráni pomocou príkazu *no ip access-list <ACL-name>.* V ďalšom kroku vykoná príkaz *show run | section interface*, ktorý vypíše informácie o konfigurácií rozhraní, v tomto prípade je predmetom záujmu iba informácia o aplikovanom ACL na danom rozhraní. Následne sú tieto ACL odstránené z rozhraní pomocou príkazu *no ip access-group <ACL-name>*. V ďalšom kroku už prichádza proces konfigurácie nových ACL, kde sa najprv vytvoria jednotlivé ACL s konkrétnymi pravidlami a následne sú aplikované na konkrétne rozhrania. V poslednom kroku sa už len vykonajú oba kontrolné príkazy, ktoré v termináli vypíšu aktuálnu konfiguráciu pre záverečnú kontrolu administrátorom. Celková doba exekúcie tohto procesu zaberie iba zopár sekúnd, takže jeho dopad na interné služby v dátovom centre je minimálny. Zároveň sa

predpokladá, že automatický proces synchronizácie bude vykonávaný bez potrebného dohľadu v noci mimo pracovných hodín, kedy je zvyčajne úroveň dátového trafficu najnižšia.

Podobný pracovný postup je využitý aj pri skriptoch vykonávajúcich aktualizáciu alebo odstránenie. S rozdielom, že úvodný výpis konfigurácie nie je v týchto inštanciách potrebný, nakoľko pre aktualizáciu je iba pôvodný ACL odstránený z konfigurácie a zároveň aj z rozhrania a následne je aplikovaný už aktualizovaný ACL. Pre odstránenie je proces ešte jednoduchší, nakoľko sa jedná iba odstránenie zvoleného ACL. V oboch prípadoch sa zobrazí v termináli aktuálna konfigurácia na zariadení aby bolo možné ju skontrolovať bez nutnosti sa pripájať na zariadenie.

```
Retrieving and clearing global ACLs...
['Allow_traffic_C-AUTOMAT']
Removing ACLs from interfaces...
Removing ACLs from interfaces on Cisco_Nexus1: ['interface FastEthernet0/0', 'no ip access-group Allow_traffic_C-AUTOMAT in',
'no ip access-group Allow_traffic_C-AUTOMAT out']
Applying new ACL configurations...
Checking final ACL configurations...
Current ACLs for Cisco_Nexus1:
Extended IP access list Allow_traffic_A-inbound
    1 permit tcp host 2.2.2.2 any
    2 permit icmp any any
    3 permit tcp any any range 32768 65535
Extended IP access list Allow_traffic_A-outbound
    1 permit udp 10.10.0.128 0.0.0.127 host 2.2.2.2 range 10 30
    2 permit icmp any any
    3 permit tcp any any range 32768 65535
Checking final ACL configurations...
Current Interfaces for Cisco_Nexus1:
interface FastEthernet0/0
 no ip address
 ip access-group Allow_traffic_A-inbound in
 ip access-group Allow_traffic_A-outbound out
```

Obrázok 6-7: Výpis v termináli popisujúci chod procesu

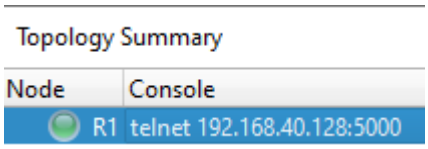
6.8. Testovanie v simulačnom prostredí

Pre simulačné účely boli použité dve rôzne simulačné prostredia a to GNS3 a EVE-NG. Obe platformy sú bežne používané pre emuláciu sieťového prostredia pre komerčné, výskumne a edukačné účely. Platformy GNS3 a EVE-NG ponúkajú podobné možnosti emulácie sieťových prvkov s menšími rozdielmi. GNS3 je bezplatný open-source software so zameraním hlavne na zariadenia od výrobcu Cisco. Ďalej ponúka pokročilejšie možnosti prispôsobenia, čo je vhodné pre komplexnejšie sieťové architektúry[60]. Na druhej strane, EVE-NG ponúka mimo bezplatnej verzie aj vylepšené

Professional vydanie s viacerými funkciami. Výhodou EVE-NG oproti GNS3 je podpora zariadení od viacerých výrobcov ako Juniper, Palo Alto, Fortinet a podobne[61]. Obe simulačné prostredia využívajú VMware Workstation Player, čo je hypervisor typu 2, ktorý ponúka jednoduché riešenie virtualizácie na desktopoch[62]. Použitie dvoch rôznych typov simulačných prostredí malo za cieľ zvýšiť spoľahlivosť výsledkov.

Postup pripojenia ide v jednoduchej sekvencii. Nornir si najprv špecifikuje konkrétne sieťové zariadenie zo súboru *hosts.yaml*, z ktorého si extrahuje IP adresu, konkrétny port a typ zariadenia. Ak sa na zariadení nachádza aj prihlasovacie meno a heslo, ktoré je potrebné zadať pri prihlasovaní na zariadenie, Nornir sa pomocou nich prihlási. Pre jednoduchosť testovacieho procesu, prihlasovacie údaje neboli využité. V ďalšom kroku sa Nornir pripája priamo na zariadenie.

Ako už bolo vysvetlené v predošlých kapitolách, Nornir využíva konzolové pripojenie na komunikáciu so zariadením. V simulačnom prostredí GNS3 bolo pre simuláciu zariadenia Cisco s operačným systémom IOS použité zariadenie z radu Cisco C7200. Konkrétne sa jedná o jeho image s názvom *7200 Software (C7200-ADVENTERPRISEK9_SNA-M)* na verzii 15.2(4)M2. Toto zariadenie plnohodnotne postačuje na overenie riešenia pre špecifický operačný systém. V prípade Cisco IOS bolo konzolové pripojenie pomocou protokolu telnet na IP adrese 192.168.40.128 a porte 5000.



Obrázok 6-8: Zariadenie Cisco IOS v simulačnom prostredí GNS3

V simulačnom prostredí EVE-NG bolo pre simuláciu zariadenia Cisco s operačným systémom NX-OS použité zariadenie z radu Cisco Nexus 9000v, čo reprezentuje virtualizovanú verziu switchu z radu Nexus 9000. Použitý image bol s verziou nxos.7.0.3.I7.2. Podobne ako v prípade Cisco IOS, konektivita bola dosiahnutá pomocou konzolového pripojenia. V tomto prípade bolo zariadenie dostupné taktiež pomocou protokolu telnet na IP adrese 192.168.218.128 a porte 32770. Je dôležité podotknúť, že pre účely testovania je nešifrovaný protokol telnet dostačujúci, ale pre premávku mimo

testovacieho prostredia je z bezpečnostného hľadiska nepostačujúci. Pri použití na fyzických zariadeniach sa preto predpokladá použitie protokolu SSH.

V poslednom kroku sa už Nornir pripojí na dané zariadenie a začne vykonávať príslušné príkazy, ktoré boli popísané v predošlej podkapitole. Na obrázku 6-6 je možné vidieť ilustračné príkazy ktoré boli zadané automaticky pomocou Norniru. Podobne vyzerá celý proces exekúcie danej konfigurácie. Testovanie bolo primárne zamerané na otestovanie konfigurácie na zariadeniach Cisco, nakoľko tvoria nadmernú väčšinu infraštruktúry. Predpokladá sa, že bezproblémové nasadenie by malo prebehnúť aj na fyzických zariadeniach, nakoľko virtuálne zariadenia zdieľajú všetky potrebné funkcionality.

7. DISKUSIA

Celkovo bolo v rámci diplomovej práce implementovaných viacero automatizačných riešení s versatilným použitím pre správu sieťovej infraštruktúry spoločnosti EmbedIT. Dôraz bol kladený na výber jednoduchých nástrojov a metód, ktoré by zjednodušili každodenné operácie sieťových administrátorov. Dôležitým faktorom bola aj užívateľská prívetivosť a intuitívnosť použitia do takej miery, aby si nové metódy nevyžadovali zdĺhavé učenie a tým zbytočne nepredĺžili celkový implementačný proces. Vďaka hierarchickej štruktúre a dobrej čitateľnosti YAML súborov môžu administrátori rýchlejšie aplikovať potrebné úpravy. Predpokladá sa, že touto integráciou sa eventuálne zníži výskyt chýb pri manuálnej práci a urýchli proces nasadzovania. S tým spojená aj implementácia centralizovaného zdroja pravdy ako jednotného bodu pre správu konfigurácií poskytuje robustnú platformu pre udržiavanie aktuálneho prehľadu o stave sieťovej infraštruktúry. Ako nadstavba riešenia zdroju pravdy je možné zakomponovať aj integráciu s kontrolnými systémami verzií. V ktorých každá zmena v súbore YAML môže byť sledovaná cez verzovacie systémy, ako je Git, čo poskytuje transparentný prehľad o histórii zmien, kto a kedy zmenil konfiguračný súbor[63]. Používanie systému Git pre správu verzií konfiguračných súborov prináša výhody ako prehľad vykonaných zmien. Git veľmi jednoducho umožňuje sledovať, kto a kedy urobil zmenu v konfigurácii. Táto schopnosť zvyšuje transparentnosť a umožňuje rýchlu identifikáciu a nápravu potenciálnych chýb alebo bezpečnostných rizík. Napríklad, ak administrátor pridá alebo odoberie pravidlo v ACL, každá takáto zmena je dokumentovaná v Gite prostredníctvom commitu. Každý commit obsahuje meno administrátora, čas a popis zmeny. Tento popis by mohol vysvetľovať, prečo bola zmena vykonaná, napríklad „Pridanie pravidla na blokovanie zakázaných IP adries“. Zakomponovanie tohto riešenia by mohlo prispieť k vyššej úrovni auditovateľnosti a zabezpečenia sieťovej infraštruktúry.

Významným prínosom zvoleného prístupu je integrácia flexibilného a široko podporovaného jazyka Python, ktorý umožňuje sieťovým administrátorom prispôsobiť automatizačné skripty aktuálnym potrebám bez zbytočného zdĺhavého učenia nových technológií. S použitím knižnice Nornir je možné efektívne a paralelne vykonávať konfiguračné zmeny na viacerých zariadeniach, čo výrazne skracuje čas potrebný na aktualizáciu a údržbu sieťového prostredia. Je možné konštatovať, že integrácia programovacieho jazyka Python a knižnice Nornir do procesov automatizácie v

spoločnosti EmbedIT predstavuje praktické a efektívne riešenie, ktoré rešpektuje obmedzenia existujúcej infraštruktúry a zároveň maximálne využíva dostupné zdroje a skúsenosti personálu. Tento prístup nielenže zefektívňuje bežné operácie, ale tiež otvára dvere pre ďalšie možnosti rozšírenia a inovácie v budúcnosti.

Celkovo riešenie pozostáva z troch hlavných skriptov, ktoré vykonávajú potrebné kroky synchronizácie, aktualizácie a odstránenia. Voči tomuto kroku je možné argumentovať, prečo nie sú aplikované iba vykonané zmeny a tak zbaviť sa nadbytočnému procesu odstraňovania a pridávania ACL. Na úrovni programovania je túto úlohu možné považovať, za časovo náročnú a zároveň nepragmatickú keďže organizácia EmbedIT využíva verzovací systém typu Git, ktorý tento postup značne uľahčuje. Nakoľko prostredie využité na programovanie a testovanie riešenia neumožňuje integráciu verzovacieho systému, bola táto možnosť po dohode so spoločnosťou EmbedIT vylúčená. Toto rozhodnutie otvára možnosti vytvorenia nadstavby tejto práce v podobe vytvorenia riešenia, ktoré by zakomponovalo použitie verzovacieho systému do architektúry, ktorý by následne jednoducho aplikoval iba zmeny od predchádzajúcej verzie. Tento krok by zároveň aj zjednodušil proces aplikácie, kedy by nebolo potrebné používať tri rôzne skripty ale iba jeden.

Ďalším možným námetom pre vylepšenie aktuálneho riešenia je pridanie možnosti spravovania viacerých nastavení v rámci rozhrania. Jednalo by sa iba o jednoduché pridanie hodnôt v definičných YAML súboroch *interfaces*, v ktorých sa momentálne nachádza iba špecifikácia aplikovaných ACL. Môžu to byť nastavenia ako IP adresa, Routing protokoly, VLAN, Quality of Service a podobne, ktoré budú nakonfigurované na zariadenie rovnakým postupom. Druhý a zároveň posledný krok tejto nadstavby by pozostával z jednoduchého pridania správnej syntaxe pre konfiguráciu na konkrétnych zariadeniach.

Celkovo je možné konštatovať, že riešenie poskytlo robustnú platformu pre možné budúce rozšírenia a inovácie. Potenciálne modifikácie je vďaka jednoduchým pythonovským skriptom možné ľahko integrovať do nových postupov v existujúcom systéme. Táto flexibilita je dôležitá pre udržanie kroku s rýchlo sa meniacim technologickým prostredím a zabezpečenie, že sieťová infraštruktúra bude schopná reagovať na nové výzvy a požiadavky.

8. ZÁVER

Diplomová práca sa zaoberala návrhom a implementáciou automatizovaného riešenia na správu sieťovej infraštruktúry v dátovom centre v spoločnosti EmbedIT. Teoretická časť práce sa venovala vysvetleniu základných konceptov automatizácie sietí a metodiky Infrastructure as Code, ktorá umožňuje hromadne spravovať infraštruktúru. Tento prístup poukázal na výhody hromadnej konfigurácie, ako sú zníženie riziko chyby ľudského faktora a zvýšenie konzistentnosti konfigurácií. Hlavná časť práce pozostávala z praktickej implementácie riešenia na automatizáciu Access Control Listov, ktoré si vyžadovali náročnú a neprehľadnú konfiguráciu. Táto implementácia bola zrealizovaná s využitím programovacieho jazyka Python a knižnice Nornir, ktoré boli zvolené s ohľadom na technické možnosti existujúcej infraštruktúry a skúsenosti administrátorov v spoločnosti EmbedIT.

Implementované riešenie má za účel signifikantne zjednodušiť prácu sieťových administrátorov tak, že eliminuje manuálne zásahy pri správe ACL. Táto automatizácia umožní rýchlu a presnú aplikáciu zmien na všetky relevantné zariadenia. Hlavná časť ďalej pozostávala z návrhu hierarchickej štruktúry zápisu použitých sieťových zariadení v topológii, ich rozhraní a použitých ACL. Týmto zápisom sa dosiahla prehľadná a systematická dokumentácia, ktorá bude slúžiť ako spoľahlivý zdroj pravdy pre požitú sieťovú infraštruktúru. Zavedenie zdroja pravdy do používanej dokumentácie spoločnosti EmbedIT pre správu ACL prispeje k maximalizovaniu konzistentnosti konfigurácie, čo nielen signifikante zníži riziko konfiguračných chýb ale zároveň aj zabezpečí efektívnejšie riešenie problémov. Použité skripty následne zo zdroja pravdy spoľahlivo vygenerujú a aplikujú tieto konfigurácie na sieťové zariadenia. Výsledné riešenie bolo otestované v simulačnom prostredí, čo umožnilo overiť funkčnosť implementácie ACL na konkrétne sieťové zariadenie. Výsledky praktickej implementácie na virtuálne sieťové zariadenia preukázali, že navrhnuté riešenie spoľahlivo a v minimálnom čase aplikovalo zadané ACL.

Na záver možno konštatovať, že riešenie tejto diplomovej práce prispeje k úspore času a nákladov. Zároveň sa predpokladá zlepšenie celkovej spoľahlivosti a bezpečnosti dátového centra. Tento prístup demonštruje ako moderné technológie a metodiky môžu transformovať tradičné postupy a priniesť výhody v každodennej prevádzke sieťovej

infraštruktúry. Diplomová práca tak poskytuje obohatenie v oblasti automatizácie sietí a môže slúžiť ako základ pre ďalšie implementácie v tejto oblasti.

LITERATÚRA

[1] Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region. *Amazon Web Services, Inc.* [online]. [vid. 2024-03-17]. Dostupné z: https://aws.amazon.com/message/41926/

[2] What is Network Automation? | Juniper Networks US. *Juniper Networks* [online]. [vid. 2024-03-17]. Dostupné z: https://www.juniper.net/us/en/research-topics/what-is-network-automation.html

[3] What Is Network Automation? *Cisco* [online]. [vid. 2024-03-17]. Dostupné z: https://www.cisco.com/c/en/us/solutions/automation/network-automation.html

[4] CATANIA, Carlos a Carlos GARCIA GARINO. Automatic network intrusion detection: Current techniques and open issues. *Computers & Electrical Engineering* [online]. 2012, **38**, 1062–1072. Dostupné z: doi:10.1016/j.compeleceng.2012.05.013

[5] NUNEZ, Alexander, Joseph AYOKA, Md Zahidul ISLAM a Pablo RUIZ. *A Brief Overview of Software-Defined Networking* [online]. 2023. Dostupné z: doi:10.48550/arXiv.2302.00165

[6] HAGARA, Michal. Evaluation of Infrastructure as a Code for Enterprise Automation [online]. 2018. Dostupné z: https://is.muni.cz/th/liwzz/

[7] MIJACOBS. *What is infrastructure as code (IaC)? - Azure DevOps* [online]. 28. listopad 2022 [vid. 2024-03-17]. Dostupné z: https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code

[8] MIJACOBS. *What is continuous delivery? - Azure DevOps* [online]. 28. listopad 2022 [vid. 2024-05-23]. Dostupné z: https://learn.microsoft.com/en-us/devops/deliver/what-is-continuous-delivery

[9] *Infrastructure as Code (IaC) Defined (with Examples) | Puppet* [online]. [vid. 2024-03-17]. Dostupné z: https://www.puppet.com/blog/what-is-infrastructure-as-code

[10] SHAHAF, Tzvika. *Agent vs. Agentless Security: Comparing Which to Use + When* [online]. [vid. 2024-03-17]. Dostupné z: https://www.puppet.com/blog/agent-vs-agentless-security

[11] MORRIS, Kief. Infrastructure as Code. *elbooksworld*. 2021, 431.

[12] What is Terraform | Terraform | HashiCorp Developer. *What is Terraform | Terraform | HashiCorp Developer* [online]. [vid. 2024-05-18]. Dostupné z: https://developer.hashicorp.com/terraform/intro

[13] *Terraform blueprints and modules for Google Cloud* [online]. [vid. 2024-05-22]. Dostupné z: https://cloud.google.com/docs/terraform/blueprints/terraform-blueprints

[14] VELICH, Ofer. Terraform vs. Ansible vs. Puppet. *Logz.io* [online]. 6. březen 2018 [vid. 2024-05-23]. Dostupné z: https://logz.io/blog/terraform-vs-ansible-vs-puppet/

[15] *Kubernetes Documentation* [online]. [vid. 2024-05-18]. Dostupné z: https://kubernetes.io/docs/home/

[16] PRANCER. Infrastructure as Code: Mutable Vs Immutable. *Medium* [online]. 30. červen 2023 [vid. 2024-05-23]. Dostupné z: https://medium.com/@prancerca111/infrastructure-as-code-mutable-vs-immutable-4cd0aab9b29d

[17] ROMETSCH, Ben. *How Ansible got started and grew | Opensource.com* [online]. [vid. 2024-03-17]. Dostupné z: https://opensource.com/article/21/2/ansible-origin-story

[18] *Ansible playbooks — Ansible Community Documentation* [online]. [vid. 2024-03-17]. Dostupné z: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html#about-playbooks

[19] HAT, Ansible, Red. *How it works* [online]. [vid. 2024-03-17]. Dostupné z: https://www.ansible.com/overview/how-ansible-works

[20] Best practices for network automation with Python | TechTarget. *Networking* [online]. [vid. 2024-03-17]. Dostupné z: https://www.techtarget.com/searchnetworking/tip/Best-practices-for-network-automation-with-Python

[21] Top 5 Programming Languages for Automation Testing. *GeeksforGeeks* [online]. 8. listopad 2022 [vid. 2024-03-17]. Dostupné z: https://www.geeksforgeeks.org/top-5-programming-languages-for-automation-testing/

[22] *Welcome to nornir's documentation! — nornir 3.3.0 documentation* [online]. [vid. 2024-03-17]. Dostupné z: https://nornir.readthedocs.io/en/latest/

[23] Ansible vs Nornir: How Python Makes Automation Easier. *CBT Nuggets* [online]. [vid. 2024-03-17]. Dostupné z: https://www.cbtnuggets.com/blog/technology/networking/ansible-vs-nornir-how-python-makes-automation-easier

[24] A Beginner's Guide to NAPALM Network Automation. *Packetswitch* [online]. 7. duben 2023 [vid. 2024-05-22]. Dostupné z: https://www.packetswitch.co.uk/napalm-network-automation/

[25] Getting Started with NAPALM. *AOS-CX* [online]. [vid. 2024-05-22]. Dostupné z: https://developer.arubanetworks.com/aruba-aoscx/docs/getting-started-with-napalm

[26] *What is Netmiko and how to use it in Python? - PyNet Labs* [online]. 24. duben 2024 [vid. 2024-05-23]. Dostupné z: https://www.pynetlabs.com/what-is-netmiko-and-how-to-use-it-in-python/

[27] *Browse Providers | Terraform Registry* [online]. [vid. 2024-05-23]. Dostupné z: https://registry.terraform.io/browse/providers

[28] Virginia, Amazon announce $35 billion data center plan. *AP News* [online]. 21. leden 2023 [vid. 2024-03-18]. Dostupné z: https://apnews.com/article/technology-data-management-and-storage-amazoncom-inc-virginia-business-c75df1f34069b09549fe15c99335b8fb

[29] The World's Top 50 Power-Hungry Data Center Markets. *OilPrice.com* [online]. [vid. 2024-05-22]. Dostupné z: https://oilprice.com/Energy/Energy-General/The-Worlds-Top-50-Power-Hungry-Data-Center-Markets.html

[30] Best Switches for Data Centers: An in-depth Analysis. *Network Devices Inc.* [online]. [vid. 2024-05-23]. Dostupné z: https://networkdevicesinc.com/community/blog/best-switches-for-data-centers

[31] What is data center architecture? | FS Community. *Knowledge* [online]. 3. březen 2022 [vid. 2024-05-23]. Dostupné z: https://community.fs.com/article/what-is-data-center-architecture.html

[32] SLAUGHTER, Jim. Leaf-Spine Deployment and Best Practices Guide. *Dell* [online]. 2017. Dostupné z: https://infohub.delltechnologies.com/static/media/a6db0ca1-66bd-46bd-a955-8f3604395191.pdf

[33] What Is a DMZ Network and Why Would You Use It? *Fortinet* [online]. [vid. 2024-03-18]. Dostupné z: https://www.fortinet.com/resources/cyberglossary/what-is-dmz

[34] How Network Segmentation Simplifies PCI DSS Compliance. *Akamai* [online]. [vid. 2024-03-18]. Dostupné z: https://www.akamai.com/blog/security/pci-dss-network-segmentation

[35] *Segmentation Strategy* [online]. 26. říjen 2020 [vid. 2024-03-18]. Dostupné z: https://community.cisco.com/t5/security-knowledge-base/segmentation-strategy/ta-p/3757424

[36] What is Access Control List (ACL)? - SearchSoftwareQuality. *Networking* [online]. [vid. 2024-05-23]. Dostupné z: https://www.techtarget.com/searchnetworking/definition/access-control-list-ACL

[37] Configure and Filter IP Access Lists. *Cisco* [online]. [vid. 2024-03-18]. Dostupné z: https://www.cisco.com/c/en/us/support/docs/security/ios-firewall/23602-confaccesslists.html

[38] *Virtual Routing Instance (VRF-Lite) | Cloud-Native Router 23.2 | Juniper Networks* [online]. [vid. 2024-03-18]. Dostupné z: https://www.juniper.net/documentation/us/en/software/cloud-native-router23.2/cloud-native-router-user/topics/concept/l3-virtual-router.html

[39] MPLS: Layer 3 VPNs Configuration Guide, Cisco IOS Release 15S - Multiprotocol BGP MPLS VPN [Support]. *Cisco* [online]. [vid. 2024-03-18]. Dostupné z: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/mp_l3_vpns/configuration/15-s/mp-l3-vpns-15-s-book/mp-bgp-mpls-vpn.html

[40] Configure VRF Leaks on IOS XE. *Cisco* [online]. [vid. 2024-03-18]. Dostupné z: https://www.cisco.com/c/en/us/support/docs/ip/ip-routing/216541-vrf-configuration-examples-on-ios-xe.html

[41] Cisco Nexus 9300-FX3 Series Switches Data Sheet. *Cisco* [online]. [vid. 2024-03-17]. Dostupné z: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-744052.html

[42] Understand the Hot Standby Router Protocol Features and Functionality. *Cisco* [online]. [vid. 2024-03-18]. Dostupné z: https://www.cisco.com/c/en/us/support/docs/ip/hot-standby-router-protocol-hsrp/9234-hsrpguidetoc.html

[43] EtherChannel in Computer Network. *GeeksforGeeks* [online]. 3. květen 2018 [vid. 2024-05-23]. Dostupné z: https://www.geeksforgeeks.org/etherchannel-in-computer-network/

[44] Key Considerations for ToR (Top of Rack) Network Switch Management | FS Community. *Knowledge* [online]. 25. červen 2023 [vid. 2024-05-23]. Dostupné z: https://community.fs.com/article/key-considerations-for-tor-top-of-rack-network-switch-management.html

[45] What is ToS (Top-of-Rack) Switching? *CBT Nuggets* [online]. [vid. 2024-05-23]. Dostupné z: https://www.cbtnuggets.com/blog/technology/networking/top-of-rack-switching

[46] What is VMware ESXi? How Does ESXi WorK? *Liquid Web* [online]. 21. červenec 2022 [vid. 2024-05-23]. Dostupné z: https://www.liquidweb.com/kb/what-is-vmware-esxi/

[47] *What is a Blade Server? FAQ | Lenovo Malaysia* [online]. [vid. 2024-03-18]. Dostupné z: https://www.lenovo.com/my/en/faqs/servers/what-is-a-blade-server/

[48] *What Is a Bare Metal Server? | IBM* [online]. [vid. 2024-03-18]. Dostupné z: https://www.ibm.com/topics/bare-metal-dedicated-servers

[49] *Check Point Quantum 16200 Security Gateway | CheckFirewalls.com* [online]. [vid. 2024-05-23]. Dostupné z: https://www.checkfirewalls.com/Quantum-16200.asp

[50] *What is YAML?* [online]. [vid. 2024-05-19]. Dostupné z: https://www.redhat.com/en/topics/automation/what-is-yaml

[51] *What Is a Network Source of Truth? - NetBox Labs* [online]. [vid. 2024-05-23]. Dostupné z: https://netboxlabs.com/blog/what-is-a-network-source-of-truth/

[52] *What is the Source of Truth, and Why is it Important? | Kyligence* [online]. [vid. 2024-05-23]. Dostupné z: https://kyligence.io/blog/what-is-the-source-of-truth-and-why-is-it-important/

[53] *Cisco Application Centric Infrastructure (ACI) Design Guide - Cisco* [online]. [vid. 2024-05-20]. Dostupné z: https://www.cisco.com/c/en/us/td/docs/dcn/whitepapers/cisco-application-centric-infrastructure-design-guide.html

[54] Ephemeral port. *Moxso* [online]. [vid. 2024-05-20]. Dostupné z: https://moxso.com/blog/glossary/ephemeral-port

[55] What Are Ephemeral Ports? *Coursera* [online]. 29. listopad 2023 [vid. 2024-05-20]. Dostupné z: https://www.coursera.org/articles/ephemeral-ports

[56] *Simplifying Ephemeral Ports with Example | GangBoard* [online]. [vid. 2024-05-23]. Dostupné z: https://www.gangboard.com/blog/simplifying-ephemeral-ports-with-example

[57] *Control traffic to subnets using network ACLs - Amazon Virtual Private Cloud* [online]. [vid. 2024-05-13]. Dostupné z: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html

[58] L, Linas. What Is a Cron Job: Understanding Cron Syntax and How to Configure Cron Jobs. *Hostinger Tutorials* [online]. 13. září 2021 [vid. 2024-05-23]. Dostupné z: https://www.hostinger.com/tutorials/cron-job

[59] Network Device Templating using Jinja and Python. *Packetswitch* [online]. 15. říjen 2022 [vid. 2024-05-23]. Dostupné z: https://www.packetswitch.co.uk/network-device-templating-using-jinja-and-python/

[60] *Getting Started with GNS3 | GNS3 Documentation* [online]. [vid. 2024-05-22]. Dostupné z: https://mother.github.io/docs/

[61] *Community Cookbook* - [online]. [vid. 2024-05-22]. Dostupné z: https://www.eve-ng.net/index.php/documentation/community-cookbook/

[62] UPDATED, Stefan Ionescu last. VMware Workstation Player. *TechRadar* [online]. 22. září 2022 [vid. 2024-05-23]. Dostupné z: https://www.techradar.com/reviews/vmware-workstation-player

[63] *Git - What is Git?* [online]. [vid. 2024-05-19]. Dostupné z: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

ZOZNAM OBRÁZKOV

ZOZNAM PRILOŽENÝCH SÚBOROV

FILES

- **acl**
  - Allow_template_X.yaml
  - **acl_generated**
- **devices**
  - Devices.yaml
  - Interfaces
    - interfaces_Nexus1.yaml
    - interfaces_Nexus2.yaml
- **scripts**
  - ACL-gen-strict.py
  - ACL-gen.py
  - Manual_All_generation.py
  - Manual_decomission.py
  - Manual_update.py
  - NORNIR_ALL.py
  - NORNIR_RUN-decomission.py
  - NORNIR_RUN-update.py
  - render-DECOMISSION.py
  - render-UPDATE.py
  - render_ALL.py
  - **nornir**
    - config.yml
    - defaults.yml
    - groups.yml
    - hosts.yml
  - **Templates**
    - acl-ALL.j2
    - acl-DECOMISSION.j2
    - acl-UPDATE.j2
  - **Temporal**