Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Automatizace nasazení a správy síťových zařízení

Plzeň 2024 Bc. Jaroslav Lehečka

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jaroslav LEHEČKA**
Osobní číslo:	**A22N0054P**
Studijní program:	**N0613A140040 Softwarové a informační systémy**
Téma práce:	**Automatizace nasazení a správy síťových zařízení**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s problematikou automatizace nasazení a provozní správy síťových zařízení (Network DevOps). Pozornost věnujte zejména existujícím open source nástrojům a řešením.
2. Navrhněte systém, který podchytí a zohlední návaznosti jednotlivých konfiguračních komponent sítě a pro jejich jednotnou správu bude používat parametrizované konfigurační šablony.
3. Dále navrhněte vhodnou skladbu nástrojů pro testování funkčnosti konfigurace v rámci vhodné CI/CD platformy.
4. Realizujte navržené systémy s důrazem na přehlednost a bezpečnost celkového řešení.
5. Připravte dostatečně reprezentativní konfigurační šablony tak, aby bylo možné celé řešení automaticky otestovat na již existující virtualizované síti skládající se z Cisco zařízení.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Martin Šimek, Ph.D.**
Centrum informatizace a výpočetní techniky

Datum zadání diplomové práce: **8. září 2023**
Termín odevzdání diplomové práce: **16. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2023

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2024

Bc. Jaroslav Lehečka

Abstract

The diploma thesis aims to simplify and standardize more complex ones an operation that is carried out following the situational awareness of individual elements and their configurations. The work is focused on individual operations associated with editing interfaces and VLAN management on the network. The parameters for performing the operations are entered into input files that describe the desired target state. Several types of operations were implemented and preparation for further expansion, a set of scripts was created for testing network security, automation of data preparation, conversion, uploading, preparation of structures for binding preservation, component binding analyzer and extensions TTP templating engine for more templates. The result is a fully functional application tested on a virtualized test environment using GNS3.

Abstrakt

Diplomová práce si klade za cíl zjednodušovat a standardizovat složitější operace při nasazování a automatizaci správy sítě, které se provádí v návaznosti na situační povědomí o jednotlivých prvcích a jejich konfiguracích. Práce je zaměřená na jednotlivé operace spojené s úpravou interfaců a správou VLAN na síti. Parametry pro provádění operací jsou zaneseny do vstupních souborů, které popisují požadovaný cílový stav. Bylo realizováno více druhů operací a příprava pro další rozšíření, byla vytvořena sada skriptů pro testování bezpečnosti sítě, automatizaci přípravy dat, konverze, nahrávání, příprava struktur pro uchovávání vazeb, analyzátor vazeb mezi jednotlivými komponenty a rozšíření templatovací engine TTP o další šablony. Výsledkem je zcela funkční aplikace otestovaná na virtualizovaném testovacím prostředí pomocí GNS3.

Poděkování

Rád bych touto cestou poděkoval Ing. Martinu Šimkovi, PhD., vedoucímu mé diplomové práce. Jeho vedení, odborné rady a neustálá podpora byly klíčové pro úspěšné dokončení této práce.

Obsah

1 Úvod

Dnešní člověk žije v poměrně sofistikované době. Mnoho zařízení, institucí a procesů běžného života funguje za pomoci informačních systémů a principů používaných v rámci ICT sféry. Tyto systémy řídí náš běžný život (ovlivňují pohyb vozidel na semaforech, řídí naše elektrárny a jiné). Jejich provoz se neobejde bez širokých datových sítí.

Datové sítě slouží pro komunikaci jednotlivých entit (serverů, služeb ...) mezi sebou, resp. umožňují komunikace mezi jednotlivými entitami dle předem stanovených pravidel (firewall). Díky velkému množství těchto komunikujících entit je nutné mít dostatečně rozsáhlou síť, jejíž správa je velmi složitá.

Pro správu těchto sítí je potřeba využívat sofistikovaných nástrojů, které zlepší a zjednoduší fázi operations u datových sítí (například Jerikan, Netbox). Hlavním úkolem těchto systémů je tzv. *Source of truth*, která má za cíl popsat a zachytit jednotlivé vazby mezi existujícími síťovými prvky a možnosti směřování datových packetů.

Centralizace spravování datových sítí do jednoho bodu má výhodu v jednotném a uceleném přístupu k celkové infrastruktuře. Z tohoto bodu (například Gitlab s pomocí CI/CD praktik) se může generovat a spravovat centrální *source of truth* a starat se s pomocí administrátora o centralizované nasazování a správu počítačové sítě. Realizace těchto operací může být provedena různými způsoby. Různé formáty umožní zmenšit množství restů v rámci konfigurace a nasazování jednotlivých změn v rámci sítě (zapomenutá VLAN, zbytečné konfigurace ...) a umožní sjednotit *network standard* v rámci celé sítě.

Detailnější pohled do této problematiky bude uveden dále v této práci.

2 Automatizace a počítačové sítě

V rámci této části budou představeny počítačové sítě jako oblast počítačové vědy. Budou také uvedeny používané modely počítačových sítí spolu s příslušnými síťovými prvky.

2.1 Počítačové sítě

Počítačové sítě představují množinu zařízení, která slouží pro potřeby distribuce požadavků od jednotlivých stanic ke koncovým prvkům (serverům, tiskárnám . . .).

Počítačové sítě můžeme dělit podle různých hledisek. Jedním z nich je rozdělení dle velikosti sítě:

- BAN (Body Area Network) - rozsah: centimetry. Využívá se například při sledování zdravotního stavu pacienta, senzory na těle, obecně zařízení na těle nebo v těsné tělesné blízkosti
- PAN (Personal Area Network) - rozsah: metry. Využití nalezne v domácích sítích pro připojení zařízení v kratší vzdálenosti od Wi-Fi routeru (mobily, tiskárny, notebooky) [2]
- LAN (Local Area Network) - rozsah: budova, oddělení, 1 lokalita - jednotlivá logická lokální síť. Jedná se o jednolitý objekt například účetní oddělení, ředitelství, zaměstnanci, atp. Většinou se `LAN` rozumí jednotlivý souvislý logický celek [3]
- MAN (Metropolitan Area Network) - rozsah: město, okres, kraj - rozsáhlejší síť spravovaná jednotlivými obcemi, krajem. Jedná se složitější síťovou architekturu, složitější spojení mezi subjekty, vazby mezi jednotlivými rozdělenými subjekty [8]
- WAN (Wide Area Network) - rozsah: celý Internet. Nejrozsáhlejší ze všech sítí, většina zařízení se pomyslně připojuje na `WAN`. Pojmem WAN se myslí jednotliví nadnárodní ISP (poskytovatelé internetu) propojující se dohromady v rámci subjektů označujících se jako peeringová centra [30] [28]

Internet můžeme chápat jako velikou nadnárodní interkontinentální množinu vzájemně propojených lokálních uzlů připojených k centrální mezinárodní síti (inter + network = internet). Za duchovního otce internetu je považován Tim Bernes Lee, který vymyslel projekt WWW, díky kterému může celá současná společnost naplno využívat služeb této technologie.

2.1.1 Typické uzly počítačových sítí

V rámci počítačových sítí najdeme různé prvky, které využívají služeb počítačových sítí. Každý z těchto elementů má svůj účel. Zařízení buď zpřístupňuje, přeposílá nebo konzumuje obsah vytvořený jiným zařízením. Mezi nejčastější zařízení můžeme zařadit PC, servery, notebooky, chytrá zařízení (klimatizace, topení, lednice, mrazničky, myčky, televize, automobily, CNC, kamery, smarthome appliances - klika u oken, hlavice topení, miniserver, dětské chůvy ...). Typicky se většinou jedná o koncové uzly (poskytují nebo požadují nějaká data). Ve výjimečných případech data jen přenáší (Linux WAN-LAN router).

Routery jsou zařízeními, která rozdělují sítě na jednotlivé segmenty a přeposílají data z jednoho vysílacího zařízení k druhému. Dále se může vyskytovat `hub`, nebo-li víceportový opakovač. Toto zařízení řadíme do fyzické vrstvy sítí, kdy je signál přeposílán 1:1 ze zdrojového portu na všechny ostatní porty.

2.1.2 Modely počítačové sítě

Počítačové sítě je pro lepší chápání a rozdělení funkcí vhodné rozčlenit do jednotlivých vrstev. Každá vrstva sousedí s vrstvou následující a dohromady vytváří kompletní zásobník jednotlivých funkcí, které pokrývají celou škálu funkcí (od komunikace na bázi jedniček a nul až po vysokoúrovňový pohled na problematiku z hlediska standardizace a internacionalizace přenášených struktur bez ohledu na jednotlivé programovací jazyky). [9]

2.1.3 ISO/OSI model

ISO/OSI je jedním ze základních modelů architektury počítačových sítí. Model byl vytvořen v rámci organizace ISO a využívá se jako koncepční rámec k popisu jednotlivých vrstev síťového stacku. Dále pokrývá celou škálu od nejnižšího hardwaru (bitová operace) až po nejvyšší vrstvu znázorňující jednotlivé aplikace a samotného uživatele (pomyslná vrstva L8). Celý systém je dělen do sedmi samostatných základních vrstev, které na sebe navzájem navazují. Každá vrstva je pomyslné rozhraní pro vrstvu následující. [9] [13]

- L1 - **Fyzická vrstva** - Fyzická vrstva zajišťuje převod proudu bitů na signál a nazpátek. Řadíme sem kabely a bezdrátové spoje.
- L2 - **Linková vrstva** - Linková vrstva spravuje přístup k přenosovému médiu, řídí přenos mezi zařízeními na této úrovni a v rámci jedné lokální sítě. Na této úrovni se bavím o tzv. *switchích*.
- L3 - **Síťová vrstva** - Síťová vrstva zajišťuje směřování dat mezi jednotlivými sítěmi a segmenty. Řadíme sem například routery.
- L4 - **Transportní vrstva** - Transportní vrstva zajišťuje řízení toku dat, podporu QoS (Quality of Services - kvalita služeb). Na programátorské úrovni se využívají Berkeley sockety. Pro adresaci dané služby se zde používá kombinace protokol + port.
- L5 - **Relační vrstva** - Jedná se o relační / session vrstvu. Je zde implementována podpora pro řízení relace - dialogu.
- L6 - **Prezentační vrstva** - Prezentační vrstva se stará o prezentaci dat a transformace dat z aplikačního formátu do podoby přenositelné pomocí sítě. Stejně jako relační vrstva není v současné době implementována.
- L7 - **Aplikační vrstva** - Aplikační vrstva je nejvyšší vrstvou. Je představována jednotlivými aplikacemi a jejich poskytovanými službami, které navazují na nižší vrstvy. Do této kategorie můžeme zařadit například webové prohlížeče, SCP klienty, e-mailové klienty a všechny chatovací komunikátory.[29] [32]

V roce 1998 byla publikována norma RFC 2321. Jejím cílem bylo naznačit původ problémů, které se nemusely nacházet ve výše uvedených vrstvách L1 - L7..

Norma přidává níže uvedené vrstvy:

- L8: člověk
- L9: organizace (organizační vrstva)
- L10: externality (právní vazby, finance atd.)

2.2 Automatizační nástroje

V současné době existuje celá řada nástrojů, které se starají o automatizovanou správu sítí. Každý z těchto nástrojů k problematice přistupuje jiným způsobem. Ve většině případu se vychází ze společné implementace rozhraní, které zajišťuje provádění jednotlivých operací nad specifikovanými zařízeními. Tyto technologie můžeme dále rozdělit do následujících skupin:

- agentové
- bezagentové
- pull nodes

Tyto skupiny rozlišují přístup k správě a nástrojům na jednotlivých strojích.

2.2.1 Agentové řešení

Agentové řešení přistupuje ke specifikovaným zařízením na stanoveném portu a komunikuje s centrálním nodem pomocí protokolu dané technologie. Výhodou tohoto řešení je, že není nutné mít na cílových instancích přístupný port 22, tedy protokol SSH (Secure Shell). Příkladem takové technologie je například `Puppet`.

2.2.2 Bezagentové řešení

Bezagentové řešení využívá pro komunikace se svými klienty standardní port SSH. Bezagentový přístup využívá SSH agenta pro komunikaci, tedy není nutné na daný specifikovaný node instalovat další software. Odpadá tedy nutnost správy a údržby vhodných verzí komunikátorů na obou stranách. Navíc je při komunikaci využíván dobře známý protokol a je tedy jednodušší využít jej spolu s výhodami přístupného debugování spojení (občas je protokol agentových služeb veřejnosti neznámý - tedy je closed source). Navíc komunikace přes SSH umožňuje zvýšení bezpečnosti a stability daného řešení. Příkladem tohoto přístupu je Ansible.

2.2.3 Pull nodes

Pull nodes funguje na principu stahování repositáře případně inventory v předem stanovených intervalech a jejich vykonávání. Příkladem tohoto řešení je `Ansible pull`.

2.3 Ansible

Ansible je open-source softwarový nástroj. Slouží především jako služba/nástroj k automatizaci, orchestraci či správě konfigurací v IT prostředí. Funguje primárně na principu bezagentového přístupu k cílovým nodům (povětšinou pomocí SSH - ale nechá se nastavit i jinak).

Při orchestraci IT prostředí je hlavním cílem usnadnit správu rozsáhlých informačních a počítačových systémů, zefektivnit jednotlivé kroky, zautomatizovat sled prováděných operací, standardizovat veškeré úkony spojené s daným use-casem a na závěr provádět jen úkony potřebné v daném případě (je-li již nějaká konfigurace v provozu a je to i cílový stav, nedělám tedy žádné změny). V našem případě se může jednat o velmi širokou škálu cílových zařízení nebo systémů, které mohou být obsluhovány tímto nástrojem. Pohybujeme se ve škále od nejnižších vrstev, tedy IoT zařízení, až po vrstvy vyšší, tedy switche, routery - obecně síťová zařízení - nebo také hypervizory a virtuální servery spolu s vrstvou postavenou nad cloudovými řešeními.

Předpis v rámci Ansible je psán pomocí jazyka YAML, který je navržen oproti JSONu tak, aby byl jednodušší k používání. Nástroj je velice efektivní a umožňuje škálování pro různé typy operačních systémů a obecně pro různé typy podkladových vrstev. [18]

2.3.1 Architektura

Jak již bylo řečeno, důležitou součástí Ansible je jazyk YAML. Pomocí něj jsou zadávány operace, které jsou následně vykonávány na cílových systémech.

Ansible dle definice rozlišuje 2 základní typy cílových uzlů:

- `Control node` nebo také řídící uzel - místo, odkud jsou řízeny veškeré operace
- `Managed node` nebo také spravované uzly - uzly (počítače, servery, síťová zařízení), které jsou spravovány z control node

Control node je obecně považován za bod, ze kterého jsou spouštěny jednotlivé operace, které se budou provádět na managed nodech.

Pro správné fungování Control node je zapotřebí mít nainstalovaný Python a přes službu pip mít také balíček Ansible. Control node spouští operace přes jednotlivé Ansible CLI tools.

Spouští se na počítači administrátora nebo vývojáře. Případně mohou být tyto operace spouštěny v pipeline na serveru, kde se provádí CI/CD

(Continuous Integration / Continuous Deployment) operace. Příkladem takového řešení je třeba Gitlab nebo Jenkins. Dále mohou být propojeny s pravidelným spouštěním správních operací. Jsou tedy navázány na CRON joby - v pevně stanovený čas.

Ukázka kódu 2.1: Ukázka kódu v Ansible

```
# Playbook pro zobrazeni hodin na cilovem Cisco switchi
---
- hosts: ios_devices
  gather_facts: no
  connection: local

  tasks:
    - name: IOS | Show clock
      ios_command:
        commands:
          - show clock
      register: clock

    - debug:
        var: clock.stdout_lines
```

Managed node jsou stroje, systémy nebo zařízení, které jsou spravovány z centrálního Control plane. Jedná-li se o spravovaný uzel, je důležité, aby se na něm nacházel Python a SSH server, případně jiný systém, který Ansible podporuje pro přenos příkazů na cílový server. Díky přístupu přes SSH může být využito kombinace jména a hesla a nebo přístupu pomocí klíčů. Tyto údaje se nechají nastavit jednotně pro všechna zařízení dohromady.

2.3.2 Ansible v počítačových sítích

Vhodnost Ansible pro počítačové sítě je nepopiratelná. Na internetu je možné nalézt příklady Ansible rolí používaných v nejrůznějších případech (viz [10]). S příchodem automatizace a standardizace správy se objevila celá řada nástrojů pro správu a automatizaci. Ansible byl jedním z nich. Své vlastnosti může skvěle uplatnit v oblasti konfigurace Enterprise IT.

Neméně důležitou oblastí, pro kterou se Ansible také hodil, byla i celková orchestrace. Ansible přišel s vysokoúrovňovým přístupem k provádění změn (tj. nebylo nutné psát jednotlivé příkazy). Tato funkcionalita je vhodná pro inženýry v IT, operační manažery a hlavně pro síťové administrátory. Díky modulům Cisca do Ansible není nutné řešit nízkoúrovňovou syntax pro

provádění jednotlivých úkonů například v Cisco IOS nebo Cisco NXOS. Vysokoúrovňovým přístupem navíc umožní technikům řešit operační problémy a nezabývat se jednotlivostmi.

Problematickou částí při používání síťových prvků a Ansible je příprava na nasazení, pokud se pohybujeme ve virtualizované části sítě a používáme příslušné simulátory pro práci s routery, switchi a jednotlivými síťovými komponentami. Příklad takového simulátoru může být například GNS3. Řeší se zde možné problémy overlay sítě virtualizované v rámci počítače a propojení této sítě s počítačem, na kterém je síť provozována. Dále může být vysoce problematická velikost obrazů jednotlivých routerů. Pohybuje se kolem 16 GB, které zabírá na RAM. Oproti tomu neskromnou výhodou je využití chybových hlášek při selhání úlohy. Při standardním zadávání příkazů jsou úlohy prováděny sekvenčně a zjištění problémů je zde prováděno pomocí několika dalších příkazů. Ansible rovnou zveřejní danou úlohu, která selže, spolu s přidanou hodnotu v podobě dalších údajů z telemetrie daného selhávajícího zařízení.

Vhodným se jeví užití Ansible v kombinaci s intend-based přístupem (přístup v rámci něhož jsou příkazy jasně definované - tj. pokud se na zařízení nachází něco v odlišném stavu, než je cílový chtěný stav, je to náležitě přepsáno Ansiblem na všem místech). Příkladem takového využití je zero-touch provisioning. Zero touch provisioning je technologie, která bere z centrálního zdroje dat informace o konfiguraci a nastavení jednotlivých rozhraní a jednotlivých detailů konfigurace. Tato data nahrává v jednotlivých okamžicích na jednotlivá zařízení. Konfigurace zařízení je následně distribuována z jednoho zdroje. Navíc nám výpadek tohoto zdroje nevadí, protože největší možný problém je neprovedení nahrání konfigurace na cílové prvky.

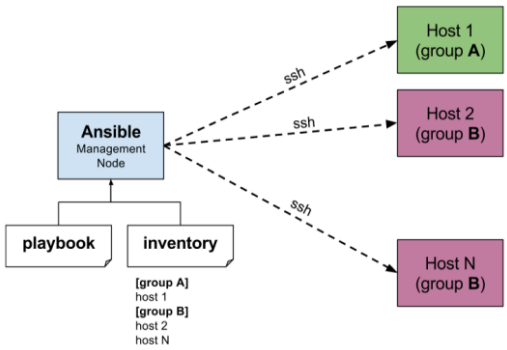
Dalším případem užití je hromadné nasazení oprav softwaru jednotlivých switchů. Každému výrobci switchů se po letech stane situaci, kdy je nahlášena chyba v softwaru. Jedná-li se o specializovaný software nebo firmware, může se stát, že pokrytí touto chybou může být velmi vysoké. V těchto případech se hodí mít nástroje, které umožní velmi rychle použít patche pro jednotlivá zařízení a omezit případně škody, které by vznikly zneužitím těchto objevených zranitelností.

Ansible pro práci se síťovými prvky používá knihoven a technologií dostupných v programovacím jazyce Python. Příkladem knihoven, které se nechají použít samostatně, jsou `netmiko`, `paramiko` a nebo `pyATS`. Ansible si vybral pro implementaci knihovny netmiko a paramiko. Na nízkoúrovňové funkce se pro SSH komunikaci používá knihovna netmiko. Pro vyšší úroveň abstrakce je použita knihovna paramiko, která umí fungovat nejen nad protokolem SSH, ale také nad Telnetem. Tyto technologie umožní Ansible

provádět vysokoúrovňový přístup na zařízení pod operačními systémy Cisca. Interfacem v Ansible jsou moduly typu `command`, `shell`, `script` a další.

Použití proměnných uložených v group_vars umožňuje uchování proměnných s konfiguracemi pro jednotlivá zařízení. Mohou zde být také uvedeny jednotlivé konfigurační parametry pro jednotlivá zařízení. Tyto konfigurace mohou být následně i upravovány v rámci playbooků. Důležité je však zachovat idempotentnost operací. Ke konfiguracím je následně přistupováno jako k poli, tj. získávání jednotlivých konfiguračních parametrů není tak náročné.

Ansible je používán spolu s různými Source-of-truth (technologie, ze kterých se čerpají data) pro nastavování cílových zařízení. Příkladem těchto technologií může být Netbox nebo v oblasti počítačových sítí technologie Jerikan. [6]



Obrázek 2.1: Architektura Ansible, Zdroj: přednášky CI/CD

2.3.3 Jednotlivé komponenty Ansible

Pro správné fungování systému je potřeba mít několik součástí a souborů. Každá ze součástí slouží jako evidence příkazů, strojů nebo proměnných.

Nepostradatelnou částí pro fungování systému je sada nástrojů pro uchovávání informací o serverech - Inventory.

Inventory

Inventory je inventář souvislých informací, který slouží pro ukládání metainformací o jednotlivých routerech, serverech a switchích. V rámci inventory může být uvedeno více informací - například pojmenování, skupina, případné přístupové údaje (uživatelská jména, hesla) nebo i další parametry příkazové řádky. Konstrukce samotného souboru může být ve formátu INI nebo YAML.

Ukázka kódu 2.2: Ukázkové INI inventory v Ansible

```
[routers]
CIV ansible_host=192.168.10.12 ansible_ssh_common_args="
    ↪ -oKexAlgorithms=+diffie-hellman-group1-sha1 -
    ↪ oCiphers=aes256-cbc" ansible_ssh_user=test
    ↪ ansible_ssh_pass=test
HLR ansible_host=192.168.10.14 ansible_ssh_common_args="
    ↪ -oKexAlgorithms=+diffie-hellman-group1-sha1 -
    ↪ oCiphers=aes256-cbc" ansible_ssh_user=test
    ↪ ansible_ssh_pass=test
[switches]
UN-A ansible_host=192.168.10.11 ansible_ssh_common_args=
    ↪ "-oKexAlgorithms=+diffie-hellman-group1-sha1 -
    ↪ oCiphers=aes256-cbc" ansible_ssh_user=test
    ↪ ansible_ssh_pass=test
[bory:children]
routers
switches
```

V ukázce kódu 2.2 je uvedený ukázkový INI inventory v Ansible, který má v sobě jednu hlavní skupinu `bory`. V rámci skupiny `bory` jsou 2 podskupiny `routers` a `switches`. Tyto skupiny dále obsahují jednotlivé prvky, které mohou být referencovány v rámci jednotlivých playbooků promocí jejich jmen (například UN-A). U každého záznamu jsou uvedeny důležité údaje pro přístup přes příkazovou řádku na SSH (IP adresa, parametry příkazové řádky, Ansible uživatel a jeho heslo). Za IP adresu zařízení může být vložen i hostname pro daný prvek.

V následující ukázce kódu 2.3 je uvedený způsob, jak provést přiřazení Ansible uživatele pro všechny prvky.

Ukázka kódu 2.3: Ukázkové INI inventory v Ansible

```
[main_node]
147.228.173.69

[all:vars]
ansible_user=nodeadm
```

Dynamické inventory

Inventář jednotlivých strojů může být buď dynamický nebo statický. V případě, že se infrastruktura mění nebo vytváří, je potřeba následně vygenero-

vat příslušné inventory z dat o vytvořených strojích. Tuto činnost můžeme například zautomatizovat v nástroji Terraform pomocí skriptů.

V případě velkých změn nebo při získávání informací z jiného informačního systému (Netbox) je výhodné využít `inventory plugins`, které nástroji Ansible umožní přístup ke strukturovaným informacím uloženým v těchto systémech pomocí direktiv a funkcí jazyka Python.

2.4 Terraform

Terraform je nástroj používaný pro IaC (infrastructure as a code). Umožňuje svým uživatelům definovat jak zdroje cloudové, tak i on-premise (tj. zdroje, které se nachází u mě ve firmě / ve stejné budově / ve stejné podsíti). Pro popis je použita relativně jednoduchá human-readable (člověkem čitelný) sada konfiguračních souborů, které definují veškeré informace pro nasazení (deploy) námi specifikovaných zdrojů (VM, velikost paměti RAM, velikost disku, operační systém - resp. obraz image operačního systému, architektura procesoru, typ providera - resp. poskytovatel cloudových služeb, připojení k internetu, počet CPU, název, ...). Ukázka je uvedena v příkladu 2.4

HashiCorp tento nástroj přizpůsobil na míru datovým centrům, proto se hlavně používá jako nástroj pro vytváření zdrojů v cloudu. Pro jednotlivé cloudové poskytovatele existuje mnoho již hotových **providery** v Terraformu pro různá cloudová řešení nebo na on-premise řešení. Pro příklad je možné uvést známé cloudové poskytovatele - Amazon AWS, Microsoft Azure cloud, GCP - Google Cloud Platform, OpenStack, OpenNebula nebo VMWare vSphere.

Mohou být vytvářeny nejen virtuální stroje, ale i datová úložiště, lokální sítě, VPN (Virtual Private Network) koncentrátory nebo dokonce virtuální Kubernetes clustery. Alternativou k tomuto přístupu je tvorba všech zdrojů na clusterech cloudových poskytovatelů ručně nebo přes CLI konzoli (Command Line Interface). Díky automatizovanému a testovatelnému přístupu je možné aplikovat a kontrolovat nastavení a standardizovat jej. Následně je jednodušší stanovovat politiky a pravidla na tvorbu jednotlivých zdrojů - resources.

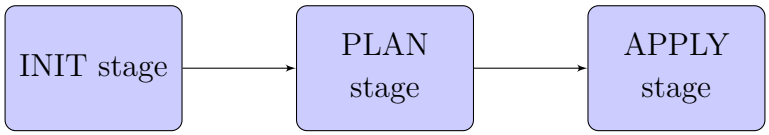
2.4.1 Architektura

Terraform je založen na několika klíčových komponentách, které pomocí hlavního konfiguračního souboru (soubor s příponou .tf) vytváří celkovou infrastrukturu podle deklarativního předpisu. Předpis umožňuje definovat,

distribuovat, upravovat, odebírat a spravovat infrastrukturu. Hlavní předností je možnost veškeré parametry a konfigurace definovat jako kód. Infrastruktura je následně verzovatelná (umožní vytvořit i rollback), testovatelná, automatizovatelná (mohu spouštět z CI/CD pipeline) a parametrizovatelná.

2.4.2 Terraform workflow

Mezi základní komponenty patří Terraform Core. Tento kompilovaný soubor je napsaný v jazyce Go. Je také hlavním blokem Terraformu. Hlavní komponenty Terraformu jsou zahrnuty v jednom kompletním souboru, který je možné přenášet z počítače na počítač bez nutni podstupovat instalaci. Po spuštění nastávají jednotlivé fáze Terraform cyklu:



Obrázek 2.2: Schéma systému

Na začátku je spuštěna **INIT** stage. Při spuštění je provedena analýza uživatelem nadefinovaných potřebných providerů. Ověří se jejich existence a existence verze. Provider se následně stáhne do složky `.terraform/providers`. V této složce jsou postupně uloženy podle zdroje (úložiště těchto provider - označujeme je registry - největší je registry.terraform.io). Dále následují složky pro verze (například 1.4.0). Následovány jsou architekturou spouštěného systému (například linux_amd64).

Ukázka kódu 2.4: Ukázka definice provider

```
provider "opennebula" {
  endpoint      = "${var.one_endpoint}"
  username      = "${var.one_username}"
  password      = "${var.one_password}"
}
```

V předchozí části již byla zmíněna podpora pro mnoho cloudových poskytovatelů. Terraform je modulárním nástrojem, tedy je možné jej rozšířit i o další nebo vlastní - naprogramované části. Je možné také spolupracovat například s technologií Docker, Kubernetes (minikube) nebo jinými nástroji, které jsou čistě na počítači, který spouští Terraform. Komunikace s těmito systémy většinou probíhá pomocí API (REST API, GraphQL API) a příslušných komunikačních protokolů. Tyto moduly jsou reálně pluginy do Terra-

formu. Plugin může být vyvíjen separátně a být aktualizován a verzován. Následně je možné jej vypublikovat na Terraform registry, které najdeme nejen v Gitlabu, ale i v Githubu. Při podpoře většiny cloudů a prostředí má tedy architekturu podobnou cloud-agnostic architektuře. Cloud-agnostic architektura má za cíl vytvořit aplikaci, která může být provozována u všech cloudových poskytovatelů nebo s nimi komunikovat.

2.4.3 State management

Terraform udržuje stav již nasazené infrastruktury v tzv. state file - stavový soubor - soubor má následně příponu **.tfstate**. Tento soubor má strukturu JSON soboru.

V rámci parametrů tohoto souboru není možné nalézt veškeré údaje, které bychom očekávali. Parametry závislé na běhu procesu se zjistí a dopíší až po doběhnutí celého procesu - příkladem takového procesu může být IP adresa. Díky tomu nástroj sleduje a tvoří seznam potřebných změn a aktualizuje jej pouze na místech definovaných v konfiguračních souborech. Některé položky se ale provádí jen při prvním nasazování - příkladem je provisioner pro soubor a pro remote-exec. Musí se počítat i s možnými důsledky našich změn. Změníme-li velikost disku (zmenšíme kapacitu disku), může se stát, že přijdeme o část dat.

2.5 Netbox

Netbox je komplexní open-source technologickou platformou pro management infrastruktury, assets managemnet a logickou správu IT prostředků, technologií, dokumentů, instancí a řízení v organizacích zaměřených na správu velkého množství těchto prostředků. Technologie nabízí rozsáhlou paletu funkcí z oblasti managementu, assets managementu, dokumentace, provisioningu přes moduly, sledování metrik, správu prvků v rámci datových center - logických a technologických - a dalších nezbytných potřeb spojených s datovými centry. Technologie najde využití v organizacích ke zlepšení a usnadnění efektivní správy zařízení, bezpečnostních pravidel a incidentů a komplexní i kompletní správy infrastruktury s tím spojené.

2.5.1 Assets management

Netbox je komplexním nástrojem pro sledování a dokumentování celé řady faktů spojených s provozem a údržbou síťové infrastruktury. Při provozu

větších technologických a logických celků je kvalitní a detailní dokumentace nezbytným prostředkem pro správný a bezpečný chod infrastruktury s nejvyšší kvalitou služeb.

Netbox dovoluje uživatelům pohlížet na infrastrukturu (a dokumentovat) z několik různých pohledů. Systém uchovává rozsáhlé informace o zařízeních (výrobce, region, lokace, IP adresy, popis, služby, obrazy, informace o kontaktech, žurnál změn, ...). Další možností je uchovávání informací o kartových routrech, které mají v sobě další moduly. Informace o nich mohou být udržovány v rámci Netboxu (např.: pro Cisco 7200). Záznamy uchovávají pro-vazby na *Module bay* a vazby na samotný *Device*.

Veškeré provedené akce jsou do detailu zaznamenávány v rámci tzv. `žurnálu`. Žurnál uchovává veškeré záznamy o provedených operacích a změnách vazeb nad specifikovanými zařízeními. Pomocí žurnálu můžeme sledovat tok změn, případně dohledávat a analyzovat chybná rozhodnutí nebo DoS útok (Denial of services) a zneužití přístupů pro útok na infrastrukturu. Tato funkce usnadňuje sledování a auditování změn v infrastruktuře.

Neméně důležitou vlastností tohoto softwaru (obzvláště v kontextu této diplomové práce) je možnost sledování závislostí mezi prvky. Vazby mezi prvky jsou obzvláště důležité z pohledu typu vazby, propustnosti, stability, specifikovaných omezení (průchod VLAN) a z hlediska identičností a návazností mezi prvky. Provedené změny je nutné mít z pohledu síťové konfigurace typicky identické na obou koncových stranách. Důležitá je také konfigurace a propustnost provozu prvků mezi těmito uzly - změny je tedy potřebné provádět nejen na jedné straně, ale i na straně druhé. Tato vlastnost umožňuje chápání a porozumění celkové architektuře a provozu.

2.5.2 Správa IP adres a DNS záznamů

Každá větší síť se neobejde bez centrální správy IP Adres a DNS. Tato funkce je klíčovým aspektem provozu IT infrastruktury. Umožňuje správcům přistupovat k jednotlivým zařízením pod jejich DNS jménem (na rozdíl od jejich IP adresy). To umožňuje jejich jednodušší správu. Správce si následně nemusí pamatovat větší množství IP adres, ale vystačí si se znalostí přeložených jmen jednotlivých zařízení. Netbox umožňuje centrální správu IP adres, kde je možné dohledat, která adresa je využita na jakých zařízeních, jaké IP adresy nám chybí a sledovat efektivní přidělování adres k jednotlivým instancím. Dále také umožňuje přidělovat DNS záznamy pro jednotlivá zařízení.

2.5.3 Sledování výkonu a integrace

Při provozu větší infrastruktury je mnohdy obtížné sledovat veškeré naměřené hodnoty ze sond na serverech, UPS apod. Pro lepší přehled, snadnější údržbu, kvalitní služby a včasnou reakci je nutné mít vybudovanou větší infrastrukturu pro monitoring a sledování výkonu.

Monitoring se nám postará o sledování výkonu, zatížení a průchodnosti jednotlivých zařízení (např. přetížený port na switchi). Na tyto situace je důležité vhodným způsobem reagovat a použít předem připravené automatizované úkony, například přidání a zvýšení hardwaru. Systém může fungovat samostatně, ale ještě častěji komunikuje s jinými zařízeními a sondami uvnitř sítě i mimo ni.

Pro komunikace používá REST API. Stejně jako zmiňované sondy může být i Netbox zdrojem dat pro ostatní systémy. Proto je nad tímto systémem vybudované REST API rozhraní, které umožňuje měnit uložené informace a posílat je do jiným systémů.

Tato integrace umožňuje větší propojení jednotlivých komponent do systémů podobných lidské společnosti. Tato propojení s existujícími řešeními vytvoří komplexní ekosystém pro správu infrastruktury.

2.5.4 Výhody používání Netboxu

Netbox je centralizovaný nástroj pro správu všech prvků infrastruktury. Tato vlastnost umožňuje a usnadňuje sledování a správu jednotlivých komponent. Díky tomu je ideálním nástrojem pro centralizované source-of-truth, který může poskytovat cenné informace pro ostatní nástroje spolu s automatizačními úkoly.

Díky centralizaci a vysokému pokrytí informací o infrastruktuře umožňuje rozumně dokumentovat, auditovat a managovat celkovou infrastrukturu. Může být i vhodným zdrojem informací pro hledání chyb a možných problémů. Toto množství informací lze využít například při hledání chybných zařízení.

IP adresy jsou využívány pro adresování dalších komponent. Je jedno, bavíme-li se o IPv4 nebo IPv6. Systém umožňuje podchycení veškerých komunikačních i adresních vazeb. Tato vlastnost může být vhodná jako zdroj informací pro automatizované úkoly s pomocí nástrojů jako je například Ansible.

Díky kompletní adresaci mohou být komplexní přehledy informací spouštěny rutinními akcemi na automatizované úrovni, které umožní spouštět ve specifických situacích příslušné opravné nebo dokumentační úkony. Rizikem

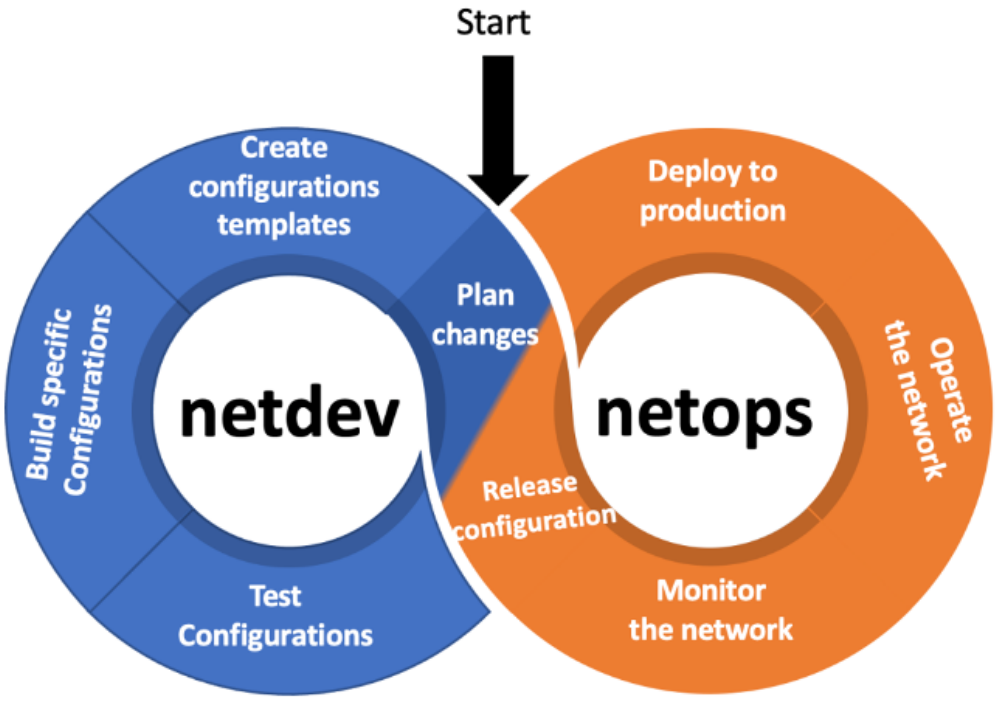
systému je neoprávněný přístup a desynchronizace dat mezi reálným systémem a Netboxem.

2.6 Přístupy k automatizaci sítě

Výhody jednotlivých přístupů pro správu sítí jsou nepopiratelné. V rámci celkové automatizace a správy sítí je nutné však zakomponovat celek do větších organizačních struktur. Proto je vhodné využít následující hotová řešení.

2.6.1 NetOps

Principy, se kterými se v současné době setkáváme a které kombinují několik oblastí v IT, se nazývají obecně XOps (například DevOps - vývojáři + operations). V případě počítačových sítí je možné se s tímto trendem také setkat. NetOps (network + operation) je trend současné doby, který popisuje celý životní cyklus projektu zaměřeného speciálně na oblast počítačových sítí a udržování sítí v chodu. Životní cyklus vývoje naznačuje následující diagram 4.4. Jde o neustále se opakující cyklus.



Obrázek 2.3: NetOps, zdroj: [4]

Při zakládání projektu je důležité plánování, které přispěje k lepší organizaci samotné práce. Následuje připrava řízení správy testování nebo-li *Test Configuration*. Správa testového řízení pomocí smoke testů kontroluje stav infrastruktury a vyhodnocuje stav operací a změn prováděných v rámci NetOps. Následně je připravená sada testů pro ověření správného fungování nasazování a správy infrastruktury. Následována je vytvářením specifických konfigurací (Build specific configuration). V této fázi se vytváří jednotlivé konfigurace popisující cílový stav infrastruktury a celkové změnové řízení. Výsledek vstupuje do další fáze (Configuration template), ve které se vytváří jednotlivé šablony, pomocí kterých je prováděna celá orchestrace infrastruktury. Následně je nutné provádění veškerých změn na infrastruktuře naplánovat na nejvhodnější možnou dobu (Plan changes). Většinou se změny provádí v nočních hodinách nebo v hodinách, kdy je infrastruktura málo využívaná nebo zcela nevyužitá. Sníží se tím dopad těchto změna na uživatele cílových produktů. Možné je i tyto akce naplánovat a provádět automaticky ve specifikovaných časech. Příkladem takového nástroje je **CRON**.

Následuje část, která se přímo týká **NetOps**. Připravené šablony jsou verzovány pomoci verzovacích nástrojů (Git, SVN), které umožňují release těchto šablon. Výsledný release je následně aplikován na cílovou infrastrukturu. To je řešeno fází **Release configuration**. Následuje další fáze, ve které je monitorováno a kontrolováno fungování a možnost provedení těchto změn na cílových nodech. Je důležité zachovat v těchto případech stabilitu řešení. Je-li jisté, že není možné provést operaci na jedné straně a také víme, že na druhé straně byly již aplikovány změny, musí se obě strany dostat do funkčního a konzistentního stavu (Rollback - vrácení změn). To je právě prověřeno ve fázi **Monitor the network**. Většinou jsou změny "vyzkoušeny"na demo síti, kde je ověřena možnost provádění těchto konfigurací. Fungováním a správou již nastavené sítě se zabývá další část této spirály, a to **Operate the network**. Veškeré nody, kterých se změny týkají, jsou připraveny do stavu, ve kterém mohou být změny aplikovány. Následuje poslední fáze, ve které jsou veškeré operace provedeny **Deploy to production**. Je ověřeno, že je možné provést veškeré operace. Síťová infrastruktura je plně připravená na provedení změn. Provedení těchto operací je nyní bezproblémové.

Při dalším požadavku se opět celý cyklus opakuje se všemi jeho částmi. [4]

2.6.2 Proč automatizovat síť

Důvodů, proč automatizovat síť, je celá řada. Pomocí automatizace můžeme efektivně spravovat a udržovat síť.

Při větším stupni automatizace můžeme dosáhnout toho, že se síť sama udržuje. Při spojení s technologií CI/CD můžeme škálovat a měnit infrastrukturu v celé její šíři nebo na jednom jediném zařízení. Počítačové sítě na rozdíl od ostatních oborů (data driven approach) v IT jsou řízeny pomocí CLI interface, které nám umožňují získávání a zadávání konfigurací do jednotlivých systémů.

Výhodou automatizace je možnost přípravy úkolů pro jednotlivé situace (event-driven engineering) a jejich automatické či poloautomatické spouštění pomocí výše zmíněných nástrojů CI/CD.

Při vysoké zátěží infrastruktury je nutné včasně a efektivně reagovat na nastalou situaci. Při spojení monitoringu a příslušných úkonů dokáže být infrastruktura připravená na vyšší nápor požadavků, víme-li, kdy máme očekávat zvýšený nápor a v jaké síle. Je důležité také rozpoznávat validní požadavky od nevalidních - myšleno DoS a DDoS útoků - tedy omezení množství požadavků na určité časové období a námi předpokládanou sílu zatížení.

Automatizační technologie nám v těchto případech pomohou připravit standardizované prostředí včas (systém ví, kolik času je nutné na přípravu škálovacích technologií - například automatické škálování databází v Azure má reakční dobu 1 minuty), v dostatečné míře (například naškáluje o 2 databázové instance navíc) a v standardizované a zabezpečené formě.

Znát, porozumět a podchytit veškeré úkony je mnohdy velmi složité. Snadno se stane, že některý klíčový úkon je zapomenut. Často automatizace pomáhá ve standardizaci a přesné replikovatelnosti postupů v jednotlivých námi definovaných případech.

Komplexnost infrastruktury při nasazování aplikací je velmi složitá. Vazby na QoS (Quality of Services), Virtual Private Network, identity management mohou být náročné na nastavení a automatizace může pomoci s těmito nelehkými kroky. [5]

2.7 YANG model

YANG je zkratkou pro Yet Another Next Generation model. YANG je popisný jazyk používaný v oblasti správy sítí a konfigurací zařízení v telekomunikačních a informačních technologiích. YANG model popisuje a definuje datové struktury a operace, které umožňují předdefinovaný a standardizovaný datový model pro udržování správy konfigurací a příslušných implementací. YANG model se například využívá uvnitř zařízení Cisco pro vnitřní uchování a implementaci datového modelu konfigurace daného zařízení. YANG

model umožňuje tvorbu modulů a +submodulů. YANG model je modulárně organizován tak, že jednotlivé moduly obsahují definované jednotlivé datové typy, skupiny a operace pro danou oblast. Pro lepší rozlišení struktury a organizace v rámci modelu byly zavedeny tzv. submoduly, které umožňují hierarchicky dělit celý model na jednotlivé části. Důležitou částí modelů jsou datové typy. YANG definuje různé datové typy (například je možné použít řetězce, čísla, identifikátory a další). Datové typy jsou důležitou součástí modelu pro popis jednotlivých struktur a obsahu dat v modelech.

YANG umožňuje modelovat jak konfigurační, tak status data, RPC a notifikace v rámci síťové komunikace. V rámci implementací je kombinován s některými management protokoly. Nejznámějšími z nich jsou NETCONF a RESTCONF. Díky probíhající standardizaci YANGu se model postupně uplatňuje i v průmyslu. Stává se tak hlavním popisným jazykem v této oblasti. Neméně důležitá je také integrace s jednotlivými výrobci hardware.

Při porovnání SNMP a NETCONF je možné vyzdvihnout některé vlastnosti NETCONF, resp. datové složky YANG modelu - která není standardizovaná. Data mohou obsahovat provozní a lokační údaje v takové formě, kterou je možno nalézt například u protokolu SNMP. Výhodou však je větší přehlednost a jednodušší orientace při konfiguraci tohoto řešení.

YANG je ve srovnání s SNMP modelem MIB více hierarchický, takže může být rozlišováno mezi konfigurací a status módem: Navíc umožňuje vysokou rozšiřitelnost.

Definice modelu může být provedena ve více programovacích jazycích. Z pohledu jazyka Python máme hned několik implementací YANG modelů v rámci knihoven. Mezi nejdůležitější knihovny se řadí pyyangbind (validator), pyyang a openconfig. Openconfig jako jediný má více implementací i ve více programovacích jazycích.[12]

Ukázka YANG modelu je uvedena v příloze.

V rámci modelu YANG 7.7 je možné získat informace popisující jednotlivá rozhraní (interfaces) v rámci síťových zařízení. Interfaces zde mohou být reprezentovány různými způsoby, ať už jako fyzické porty (například Ethernetové porty) nebo jako rozhraní sítí (například VLAN). Identifikace jednotlivých rozhraní probíhá pomocí jejich pojmenování, v druhé řadě pomocí popisků jednotlivých interfaceů (description).

Pro každý interface jsou definovány následující údaje:

- type: Typ rozhraní
- enabled: Boolovská hodnota reflektující, zda je rozhraní enabled/disabled

- link-up-down-trap-enable: Enum pro určení, zda jsou odesílány traps (upozornění) a informace o změně (stavu spojení)
- admin-status: Enum indentifikující administrativní status rozhraní - up / down
- oper-status: Enum indikující operating state pro interface
- last-change: Datum a čas poslední změny stavu interface
- if-index: Index pro interfaces v přehledu všech interfaců
- phys-address: Fyzická adresa pro interface
- higher-layer-if: List referencí na vyšší vrstvy rozhraní
- lower-layer-if: List referencí na nižší vrstvy rozhraní.
- speed: Rychlost rozhraní v bit/s.
- statistics: Statistiky síťových dat prošlých daným interfacem, jako jsou počty přijatých a odeslaných packetů a bytů, počet zahozených packetů a ERROR stavů.

2.8 Policy decision and enforcement points

V rámci řízení, ověřování a udělování přístupů je možné nalézt dva základní koncepty. Těmito koncepty jsou Policy Decision Point (PDP) a také Policy Enforcement Point (PEP).

Policy Decision Point je komponentou, v rámci níž probíhá rozhodovací proces mezi stranou žádající (náš program) a Policy enforcement pointem. Cílem je vyhodnocovat na základě stanovených rozhodovacích pravidel (decision policies) o vstupující entitě ke zdroji, jejích metadatech a k možnosti vyhodnocení na Policy enforcement pointu. Důležité je dané rozhodnutí o udělení či neudělení přístupu ke službě či zdroji, které je prováděno dále. Policy decision point obrdží na začátku požadavek na autorizaci od jiné komponenty obecného systému a následně je prováděno vyhodnocování v rámci námi definovaných procesních postupů, politik a pravidel, které jsou pro rozhodování poskytnuty a jeví se jako potřebné. Rozhodnutí o udělení či splnění daných pravidel jsou následně zaslána dotazovateli.

Policy enforcement point je silně navázána na komponentu Policy decision point. Tento bod se přímo stará o samotnou kontrolu splnění vstupních pravidel a monitorování provádění těchto politik. Hlavním cílem tohoto bodu

je zajištění a dodržování politik a pravidel námi zadefinovaných v systému. Rozhodování probíhá na základě požadavku, který je doručen z Policy decision pointu. Na základě něj a dostupných vstupních politik se následně provádí operace rozhodnutí o splnění daných politik či jejich zamítnutí. Výsledek tohoto požadavku musí být opět zaslán zpět dotazovateli.

S tímto systémem je možné se setkat v různých oblastech IT. Od síťových prvků, přes operační systémy, aplikace či správu síťové infrastruktury. Mezi tyto systémy můžeme ze světa open-source zařadit například OPA - Open policy agent. Tento nástroj může být relativně snadno kombinován s IaC nástrojem Terraform. Rozhodnutí probíhá na základě plánu, který provádí Terraform (*.tfplan). Je rozhodnuto například o minimálním přidělení paměti pro námi definované systémy nebo přesáhnutí maximální velikosti disku na serveru. Komerční alternativou k tomuto systému může být HashiCorp Sentinel.

2.9 Testování v počítačových sítích

Transformace, konfigurace a údržba počítačových sítí není z hlediska testování moc odlišná od klasického vývoje software. Najdeme zde praktiky, které se mohou při automatizaci a konfiguraci počítačových sítí směle uplatnit (například Test Driven Development). Jak již bylo nastíněno, v počítačových sítích je možné provádět operace testování způsobem, který známe z vývoje aplikací. Testování se v počítačových sítích uplatňuje jako standard pro odchytávání chybné konfigurace či nedůsledné manipulace se zařízením. Proto se tento proces hojně zavádí ve větších sítích pro odchycení chyb a nesrovnalostí v konfiguracích.

2.9.1 OPA - Open Policy Agent

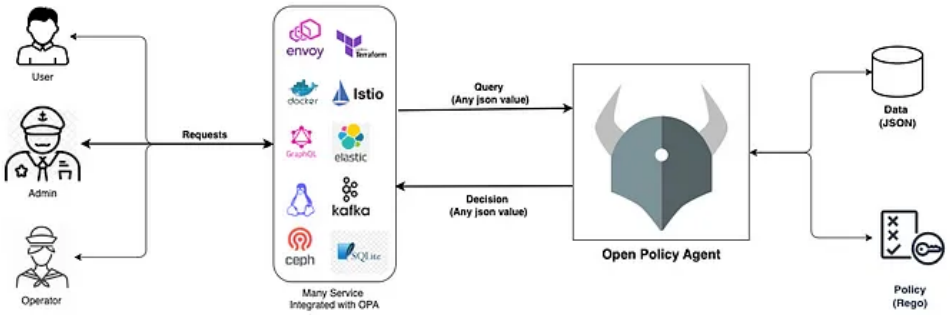
V dnešní době se prostředí počítačových sítí rychle mění. Proto je nutné u organizací prosazovat zásady, které zajistí přístupnost, bezpečnost a spolehlivost běhu operací a standardizaci úkonů při výpadku či útoku na službu. S nárůstem hybridní infrastruktury (část infrastruktury je v rámci organizaci - tj. on-premise - a část je v cloudu) je nutné prosazovat bezpečnostní pravidla a zásady napříč zde zmíněným heterogenním prostředím. Stalo se tedy náročnějším propagovat a prosazovat tato pravidla na infrastruktuře, která je distribuovanější a dynamičtější, než bylo dříve zvykem.

Příkladem nástroje, pomocí kterého může být dosaženo těchto cílů, je Open Policy Agent - OPA. Open Policy Agent je nástroj, který pomáhá organizacím se správou a prosazováním nastavených politik přes všechna

softwarová, informační a síťová řešení. Hlavním tvůrcem tohoto softwaru je společnost Styra. Mezi pravidla, která je třeba v organizaci vynucovat, můžeme zařadit bezpečnostní, správní nebo regulační politiky.

OPA poskytuje neocenitelný nástroj a přístup pro vývojáře pro psaní pravidel (policies) v jazyce Rego. Rego nabízí jednoduše využitelný jazyk pro vývoj a správu pravidel. Jazyk je uspořádán do souborů. Výsledný obsah těchto souborů je human-readable - tj. člověkem čitelný. Výhodou textové reprezentace je možnost využít Git úložiště pro tato pravidla. Takže je možné je následně i verzovat. Případně se i v rámci jednotlivých verzí pravidel vracet na předchozí řešení.

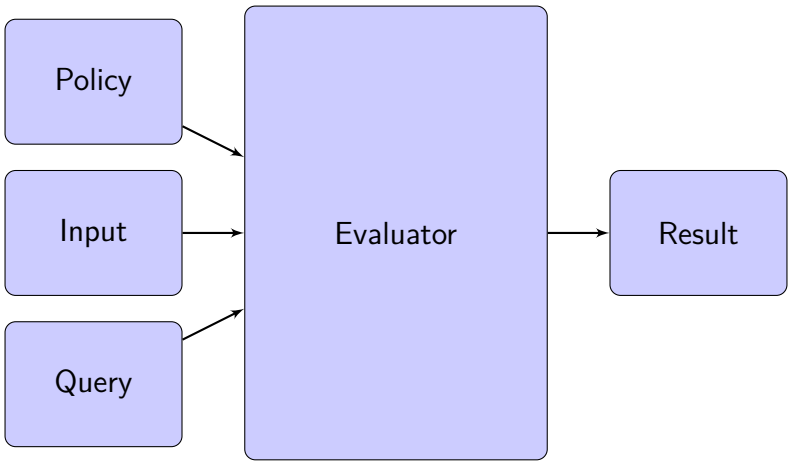
Výhodou řešení OPA je vysoké množství integrací do různých softwarových systémů. Integrace je tedy prováděna nejen na úrovni programovacího jazyka, ale i v rámci informačních systémů. Mezi podporované systémy patří například envoy, docker, terraform, elastic search, SQLite, Linux nebo také úložiště Ceph.[20]



Obrázek 2.4: Open Policy Agent, zdroj: [20]

Princip fungování Open Policy agenta můžeme popsat jako postup jednotlivých operací, který vede buď k zamítnutí nebo k potvrzení. Open Policy Agent získá na vstupu pravidla, vstup a vstupní požadavek a na základě toho rozhoduje o odpovědi. Na vstupu může být víceméně jakákoliv validní JSON (JavaScript Object Notation) struktura. Tuto strukturu je možné vytvořit fakticky v jakémkoliv programovacím jazyce. Propojitelnost je tedy platformně a jazykově (z hlediska programovacího jazyka) nezávislá. Vizualizace schématu fungování systému je zanesena do diagramu 3.2. [21]

Důvodů, proč používat Policy enforcement point a Policy decision point v kombinaci s Open Policy Agentem, je mnoho. Open Policy Agent je open-source nástrojem - jako jeden z mála nástrojů v této skupině. Open Policy



Obrázek 2.5: Diagram toku dat

Agent umožňuje, na rozdíl od některých jiných komerčních nástrojů, definovat pravidla pomocí přístupu Policy as code. Ostatní nástroje nevyužívají zápis do kódu, ale veškerá správa řešení probíhá pomocí uživatelského rozhraní. Policies as code umožňuje navíc dodržovat již zavedené praktiky (standardní životní cyklus všech vyvíjených aplikací) pro vývoj softwaru ve firmách. Je možné je integrovat s Merge requesty (paralelní práce na pravidlech) a generovat pravidla strojem, CI a testy. Navíc oproti ostatním nástrojům uchovává Git historii změn, takže je možné dohledat změny, případně se vracet v pravidlech v čase. Jak již bylo zmíněno, OPA využívá JSON na vstupu, tedy je propojitelný se všemi programovacími jazyky, které tento formát podporují. Jelikož jsou již hotové integrace do různých hotových systémů, je možné námi psaná pravidla aplikovat napříč několika systémy dohromady. To umožňuje zjednodušení a zpřehlednění správy všech pravidel na jedno místo. Výhodou je i snazší auditovatelnost všech změn v pravidlech a jejich případná úprava. Díky této vlastnosti je snazší se vyvarovat nechtěných závislostí na technologiích a můstcích (propojení mezi systémy a sdílení dat mezi nimi) jiných stran. Navíc je v rámci Open Policy Agenta implementován modul pro tvorbu testů. Přímo v rámci implementace je možné využívat všech technologií, které známe z testování kódu. Hlavním bodem při testování je modul pro Unit testy. Unit testy umožňují v separátním souboru (v jazyce REGO) definovat všechny testovací případy pro daný soubor politik. Následně je možné tento modul přidat do integrační pipeline ve fázi Continuous integration na Gitlabu. Další užitečnou technologií, která se objevuje i u klasických testovacích frameworků, je nástroj pro Mockování dat - vytváření a příprava vazeb na jiné komponenty. Vše je umožněno pomocí klíčového slovíčka replace. Na závěr jsou zde klasické nástroje pro ověřování

pokrytí kódu testy a samozřejmě Profiling. Napojení existuje i pro Gitlab, kde je možné veškeré politiky a výsledky jejich testování centrálně vést. [22]

2.9.2 Ansible Molecule

Ansible je jedním z nejznámějších nástrojů pro nasazování infrastruktury. Pro standardizaci a přiblížení všech postupů z klasického vývoje byla k Ansible doprogramována řada nástrojů. Jedním z nástrojů pro testování je i Ansible Molecule. Ansible Molecule je možné využít při ověřování infrastruktury - resp. zda deployovaná infrastruktura přesně odpovídá našemu předpisu. Kontrola pak následně probíhá vůči reálné infrastruktuře. Dále nabízí základní nástroje pro linting (statická analýza kódu), idempotenci, multikontejnerovost (více kontejnerů, které jsou obsluhovány přes Ansible + Molecule) a závislosti pro backend, které umožní deploy a ověřování správné funkčnosti naprogramovaných Ansible rolí.

Ansible Molecule slouží primárně pro testování Ansible rolí. Pro potřeby testování se uměle vytváří testovací prostředí, ve kterém se buď pouští testy, nebo vůči němuž jsou spouštěny nástroje. Navíc poskytuje nástroje potřebné pro nastavování a spouštění automatizovaných testovacích scénářů. Mezi hlavní výhody Ansible Molecule patří možnost kompletní automatizace testování sítě. Jednotlivé testovací scénáře jsou definovány pomocí YAML souborů a pak jsou špouštěny automatizovaně. Podkladová technologie (může se jednat o jakýkoliv typ virtualizace nebo kontejnerizace - například Docker nebo Podman) se postará o přípravu prostředí, na kterém se naše testy spustí. Výhodou mimo automatizované spouštění je možnost vést celý systém testů v Git repozitáři, takže mohu přecházet mezi jednotlivými verzemi testů a výsledky testů následně vypublikovat do Package registry. Následně je možné jednoduše propojit testování s CI/CD pipeline, takže veškeré zveřejněné role budou automatizovaně testovány a podle námi nastavených pravidel následně vpuštěny do produkčního prostředí, případně zveřejněny v Ansible Galaxy. Ansible Molecule má širokou podporu pro mnoho poskytovatelů (Cisco, AWS, Microsoft, ...). Je tedy možné propojit naše testování spolu s testováním hybridní infrastruktury, která je využívána. Navíc každý testovací scénář je spouštěn v izolovaném prostředí. Proto výsledky jednotlivých testů nejsou ovlivněny neúspěšnými předchozími testy. Testy se spouští v samostatném kontejneru nebo případně ve virtuálním prostředí na virtualizační platformě dle našeho nastavení. Jednotlivé role se navíc při testování nemohou ovlivňovat mezi sebou. Výsledky tedy reflektují skutečnou funkčnost jednotlivých rolí. Díky automatizaci a široké propojitelnosti není problém propojit řešení na bázi Ansible Molecule se všemi dostupnými CI

nástroji jako jsou například Gitlab a Jenkins.

[14] [15] [16] [17]

2.9.3 Koncept agnostického testování

Agnostické testování počítačových sítí je jednou z nejefektivnějších a nejvíce flexibilních strategií při provádění automatizovaného testování nad virtualizovanou sítí. Tento koncept byl vytvořen komunitou testerů, která chtěla vytvořit efektivní koncept pro použití v kombinaci s automatizací. Tento přístup nabývá na významu s přibývajícími výzvami této doby. Do této kategorie výzev můžeme zařadit například privátní, hybridní, multi-providerový nebo dokonce multi-architektonický cloud computing a software defined networks. Tento přístup je postaven na schopnosti být univerzálním a reagovat i na jiné typy síťových uzlů - v našem případě by to byly jiné příkazy pro jednotlivé verze a typy operačního systému ve switchích Cisco.

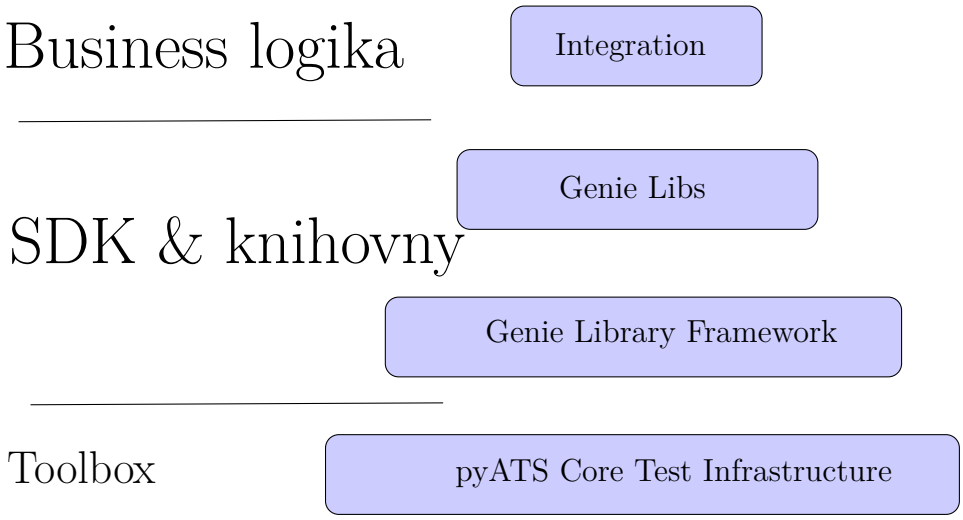
2.9.4 Cisco PyATS

PyATS byl původně interně vyvíjený program. Sloužil jako inženýrský nástroj ve společnosti Cisco. PyATS slouží jako jádro pro testování a pro TDD (test driven development). V roce 2017 se poprvé stal veřejnosti dostupný pod general public licencí.

Má následující výhody:

- vyvíjeno přímo v Cisco
- je považován za hlavní testovací framework pro inženýry vyvíjející své řešení na technologiích společnosti Cisco nebo přímo v ní
- pro všechny možné dostupné Cisco platformy, vyvíjené a používané funkce
- zakomponovatelné do CI/CD pipeline
- milionkrát využíváno celosvětově
- používá se od Sanity checků, scale, high availability a Proof-of-concept testy na denní nebo měsíční bázi
- standardizovaný nástroj pro celý cyklus testování

Při pohledu na schéma softwarové struktury systému PyATS je možné rozlišit několik základních spolupracujích komponent a vrstev.



Obrázek 2.6: Struktura PyATS

Nejvyšší vrstva je businessová logika, která obsahuje integrační vrstvu. V rámci této vrstvy je PyATS napojován na ostatní systémy a je hlouběji zapojen do workflow jednotlivých projektů. Objevuje se zde napojování na systémy podporující `Continuous integration`. Příkladem tohoto systému je například Gitlab, který byl použit i v rámci této práce. Výhodou je i možné propojení s `Continuous delivery`, tedy přímého nasazování do staging / produkčního prostředí. PyATS je schopen spolupracovat i se systémem Jenkins. Vývojáři ve společnosti Cisco vytvořili i můstky, které umožňují spolupráci se systémy jako RobotFramework.

Dalším blokem pod business logikou je **SDK a příslušné knihovny**. V rámci tohoto bloku najdeme **Genie** knihovny a frameworky.

Pod integrační vrstvou se přímo nachází knihovna Genie. Genie je nejen knihovna, ale i na testy zaměřený soubor knihoven, který umožňuje, zlepšuje a usnadňuje vývoj. Další jeho výhodou je podpora opětovného použití a zjednodušení automatizace při psaní testů. Genie je psaná v objektovém Pythonu.

Genie Library Framework je soubor knihoven, tvoří základ pro vrstvu vyšší a poskytují veškerý základ a komfort pro tvorbu jednotlivých celků. Tvoří se zde tzv. `boilerplate` a univerzální testovací příkazy označované `TestCase`, které tvoří základ pro jednotlivé testy. Najdeme zde také celou řadu parserů, modelů různých síťových protokolů. Při jednotlivých testech jsou také volány různé funkce a v předem stanovených momentech je možné volat a vyvolat speciální obslužné funkce na tyto situace. Tyto funkce označujeme `Triggery`, reps. `Handlery`. Celý Genie Library Framework je agnostický framework. Výhodou je široká napojitelnost na různé programovací ja-

zyky. Genie Library framework může být použit například v programovacím jazyce Julia. Další výhodou je možná napojitelnost na webový dashboard pro spravování (management) vstupních test suit, testbedů, výsledků testů a jejich agregací.

Úplně v nejnižší vrstvě je možné nalézt vrstvu **Toolboxu**. Tato vrstva je ze všech nejpodstatnější. V rámci této vrstvy se testovací framework připojuje na samotná zařízení. Dále také zajišťuje nastavení samotných síťových prvků před prováděním příslušných příkazů a případně i jejich vrácení do původního stavu. Také se stará o celkovou konverzi jednotlivých příkazů do formy a jazyka dané verze a typu operačního systému (například Cisco IOS verze 15.3 systému Cisco C7200). Tato část frameworku musí být schopna také podchytit veškeré informace související s topologií a převody do objektové podoby. Následně je možné aplikovat tyto testy, o jejichž spuštění se také postará. Mimo jiné musí ještě zajišťovat veškeré návaznosti na vrácení výsledků testů pro uživatele, případně vrácení detailních chyb / nesrovnalostí v případě negativního výsledku daného testu. Výsledky je nutné vracet v návaznosti na síťovou topologii a vazby daného síťového prvku, protože selhání (FAILURE) může nastat z důvodů jiných než konfiguračních. Tyto informace je samozřejmě nutné zahrnout při návrhu, přípravě a konstrukci jednotlivých testů síťové infrastruktury jako celku.

PyATS využívá konceptů tzv. „testbedů“. Tento koncept se používá pro popis jednotlivých testovaných zařízení. Pro testování je využíváno spojení se skutečným operačním systémem Cisca. Pro připojení k tomuto operačnímu systému je samozřejmě také nutné uvést dostatek informací pro příkazovou řádku (SSH klíč, případně jméno a heslo a parametry příkazové řádky pro protokol SSH a port). Jelikož se platformy od sebe liší, uvádí se také i typ operačního systému (například **iosxe**).[24]

2.10 GNS3

GNS3 (Graphical Network Simulator-3) je open-source nástroj. Slouží pro simulování počítačových sítí. Pomáhá všem uživatelům (inženýři, správci počítačových sítí, síťový architekti, DevOps technici, vývojáři a studenti) vytvářet, testovat předem nadefinované počítačové sítě a experimentovat s nimi v zabezpečeném prostředí simulátoru. Simulace počítačových sítí v simulátoru nabízí uživateli mnoho výhod při srovnání s fyzickými sítěmi. Mezi hlavní výhody patří flexibilita (finanční i časová), jednodušší škálovatelnost a možnost opakovat jednotlivé testy.[19] [11]

2.10.1 Historie

GNS3 vznikl původně jako open-source alternativa ke komerčnímu Cisco Packet Traceru. Simulátor vyvinul Jeremy Grossmannem v roce 2007. Od té doby se u odborníků stal jedním z nejpopulárnějších nástrojů pro práci se síťovou infrastrukturou: Nástroj je aktivně udržován a dále celosvětově rozvíjen (o další moduly a síťové prvky) komunitou odborníků.

2.10.2 Funkce

GNS3 poskytuje uživatelům možnost vytvářet z obrazu operačního systému síťového prvku virtuální síťová zařízení (virtualizované repliky například Cisco 7200, Cisco IOS, Cisco NX-OS) a následně je propojovat do složitých topologií. Mezi důležité funkce tohoto simulátoru je možné zařadit:

- Možnost využít platformy Docker pro virtualizaci informačních systémů, síťových prvků nebo firewallů
- Podpora široké škály virtuálních zařízení od různých výrobců
- Podpora výrobních řad síťových prvků od nejznámějších výrobců síťových zařízení: Cisco, Juniper, Arista, MikroTik RouterOS a další
- Možnost simulovat a testovat sítě s reálným provozem (nebo se simulovaným provozem) pomocí integrovaných modulů pro dynamický routing/switching a spouštění skriptů.
- Grafické uživatelské rozhraní umožňující snadné vytváření a konfiguraci sítí pomocí přetažení a poklepání.
- Integrace s cloudovými službami, jako je AWS (Amazon Web Services) a Azure, pro simulaci hybridních cloudových prostředí.
- Simulace cloudového prostředí pomocí lokálního Docker prostředí (AWS LocalStack)
- Podpora pro sdílení jednotlivých artefaktů s ostatními uživateli pomocí GNS3 Marketplace.

2.10.3 Využití

GNS3 najde široké uplatnění v různých oblastech Network engineeringu a u vývoje. Mezi jeho hlavní využití je možné zařadit:

- Výuka a školení: GNS3 poskytuje studentům a profesionálům možnost experimentovat s různými síťovými architekturami při zkoušení různých síťových technologií a realizace široké škály scénářů bez rizika poškození fyzických zařízení.
- Certifikace: Největší výrobci síťových prvků nabízí různé certifikační kurzy. Mezi nejžádanější a nejznámější je možné zařadit kurzy společnosti Cisco - Cisco CCNA a Cisco CCNP, kde GNS3 najde své uplatnění při praktickém cvičení a přípravě na tyto zkoušky.
- Výzkum a vývoj: Vývojáři síťových aplikací a služeb mohou využít GNS3 k testování svých produktů v různých prostředích a podmínkách.[7] [1]

2.11 Continuous Integration/Continuous Deployment (CI/CD)

Continuous Integration (CI) a Continuous Deployment (CD) jsou moderní vývojové praktiky, které využívají vývojáři a technici (Operations) pro rychlejší, snadnější a standardizované vydávání nových verzí softwaru. Continuous Integrations primárně cílí na pravidelné mergování (slučování) z různých vývojových větví do hlavní produkční větve a automaticky spouští různé sady testů pro ověření kvality kódu a pro pokrytí kódu testy. Continuous Deployment následně automatizuje nasazení nových verzí software do produkčního buildu nebo do nového balíčku. Případně je také možné provést nasazení aplikace do testovacího, staging nebo produkčního prostředí. Nasazení může být provedeno pomocí protokolů FTP nebo SCP. Tento přístup je považován za nejjednodušší.

2.11.1 Continuous Integration (CI)

Continuous Integration je sada operací, kterou je možné popsat pomocí následujících klíčových prvků:

- **Sledování změn u jednotlivých větví**: Vývojový tým pravidelně přidává do svých vývojových větví nové funkcionality systému a vkládá je do centrálního repozitáře (Git). Úpravy jsou následně přenášeny jen jako soubor prováděných změn.
- **Automatické testování kódu**: Při nahrání změn kódu je možné pouštět nad těmito změnami automatické testy Není to ale vždy pravi-

dlem. U větších informačních systémů je sestavování a testování drobných změn vysoce časově náročné. To klade vysoké nároky na výpočetní prostředky prostředí pro spouštění a sestavování tohoto software.

- **Ukládání výsledků testů**: Výsledky testů je možné ukládat do Package registry. Tyto soubory mohou být dále použity v jiných repozitářích pro potřeby automatizovaného analytického vyhodnocování. Tyto soubory je navíc možné stahovat a upravovat na lokálním počítači. Výsledky jsou tedy distribuovatelné po síti. Mohou být zobrazeny například u centrální obrazovky s informacemi o vývoji aplikace.
- **Kontinuální integrace**: Změny kódu jsou pravidelně integrovány do hlavní vývojové větve, což minimalizuje možnost konfliktů mezi integrovanými změnami a umožňuje navíc standardizaci a urychlení vývoje. Díky integraci je možné následně software častěji nasazovat a nasazení vykonávat po menším počtu změn. Aplikace je tedy snadněji nasaditelná a aktualizovatelná. Proces je navíc standardizován, takže je možné jej i spouštět na počítačích jednotlivých vývojářů. [31] [27] [25]

2.11.2 Continuous Deployment (CD)

Continuous Deployment je sada operací, které poskytují automatizaci pro nasazení jednotlivých verzí aplikace do předem definovaného prostředí. Prostředí může být dále rozděleno na staging, testovací, předprodukční a produkční. Tato prostředí mohou být dále předvedena zákazníkům pro otestování aplikace nebo pro potřeby penetračního testování. Další možností je využití těchto strojů pro zátěžové testování aplikace. Veškeré tyto akce je možné v rámci CD automatizovat.

Klíčové pro fungování Continuous Deployment jsou následující vlastnosti:

- **Automatizované nasazení**: Schválené změny kódu jsou automaticky nasazeny do produkčního prostředí bez manuálního zásahu.
- **Automatizace postupů**: Procesy nasazení jsou plně automatizované a opakovatelné, což snižuje riziko chyb a zvyšuje spolehlivost.
- **Správa jednotlivých verzí**: Nasazení je sledováno a řízeno pomocí správy verzí, což umožňuje jejich rychlou obnovu.
- **Monitorování výkonu**: Po nasazení jsou monitorovány výkonnostní metriky aplikace, což umožňuje identifikaci a řešení problémů v reálném čase.

2.11.3 Výhody CI/CD

CI/CD přináší řadu výhod pro vývojářské týmy a organizace:

- Zvyšuje rychlost dodávání software: Automatické testování a nasazení umožňuje rychlejší a pravidelnější dodávání nových funkcí a aktualizací.
- Zlepšuje kvalitu kódu: Pravidelné spouštění testů a automatizované nasazení pomáhají identifikovat a odstranit chyby v kódu.
- Snižuje riziko chyb: Opakovatelné procesy a automatizované postupy snižují riziko lidských chyb a zvyšují konzistenci dodávaného software.
- Zvyšuje spolehlivost aplikace: Monitorování výkonu a rychlé nasazení oprav umožňuje rychlou reakci na problémy a minimalizuje dobu výpadku.
- Posiluje agilitu a inovaci: CI/CD umožňuje agilní vývoj a rychlé přidávání nových funkcí, takže je možné velmi rychle reagovat na změny v topologii a plnit veškeré uživatelské požadavky, které má CI/CD splňovat.
- Aplikaci je možné častěji nasazovat do produkčního prostředí
- Veškeré úkoly jsou standardizované a automatizované
- Při tvorbě CI/CD pipeline je nutné zohlednit veškeré části vývoje - to má pozitivní dopad na organizaci projektu
- Centralizace všech operací do jediného místa [23]

Celkově lze říci, že CI/CD je klíčovým prvkem moderního softwarového vývoje, který pomáhá organizacím zrychlit dodávku software, zlepšit kvalitu kódu a zvýšit spolehlivost a konkurenceschopnost aplikací.

3 Analýza a návrh řešení

V rámci této kapitoly budou představeny koncepty a návrhy na řešení automatizace operací souvisejících s počítačovou sítí v souladu s teoretickou částí.

3.1 Analýza řešení

Pro potřeby vyzkoušení architektury a seznámení se s konceptem vytváření vazeb mezi síťovými prvky jsem si vybral use-case: vytváření VLAN z jednoho síťového prvku do cílové budovy. Pro vyzkoušení konceptu jsem vytvořil dummy síť. Síť je v Pythonu reprezentována pomocí grafu v balíčku NetworkX.

3.1.1 Typy vazeb a vrstev počítačové sítě

Celý úkon se skládá z několika podúkolů. Nejprve je nutné zjistit cílový prvek, ke kterému se bude v grafu hledat příslušná cesta. Tato informace je získávána z databáze organizační struktury. Z databáze je vybrán cílový prvek: budova, lokalita ... a k němu je vybrán routující device. Následně je z bodu, ze kterého chceme distribuovat cílovou VLAN, nalezena trasa k hledanému prvku. Podél celé této trasy se musí vytvořit příslušné předdefinované úkony (definované v separátní tabulce úkony). Od cílového nodu se dále hledají cílové nody, kam všude se bude distribuovat VLAN (přes celou budovu). Proto je následně aplikován nad podgrafem algoritmus DFS, který vyhledá příslušné elementy. Nad nimi se opět aplikují stejné úkony jako nad předešlými nody.

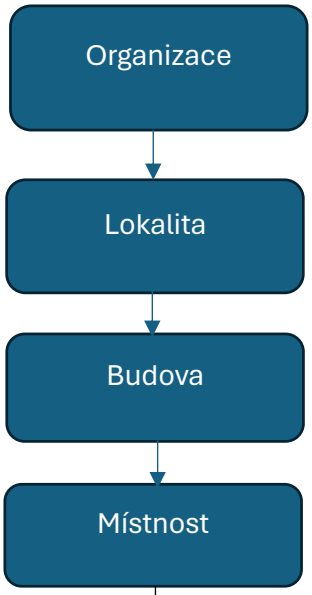
Vizualizace organizační struktury viz dále. Hierarchii počítačových sítí jsem se v konceptu rozhodl rozdělit na 3 základní úrovně.

Typy vrstev v síti:

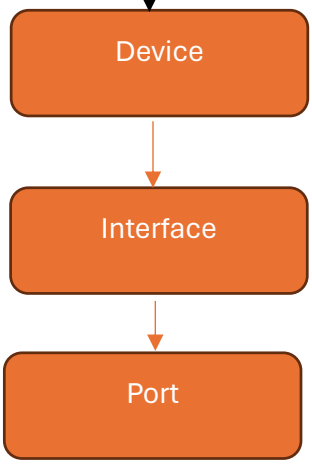
- ORGANIZAČNÍ VRSTVA
- LOGICKÁ VRSTVA
- FYZICKÁ VRSTVA

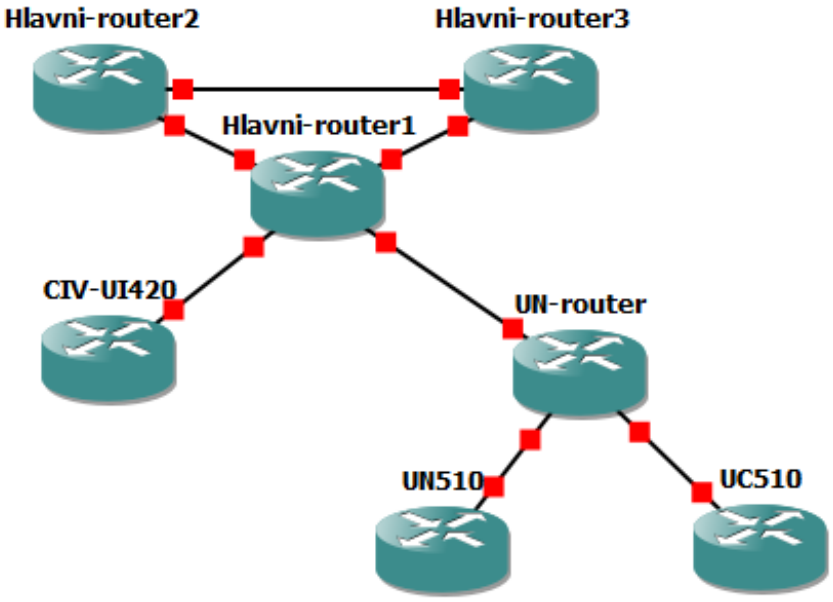
Tyto vrstvy se navzájem překrývají. Každá má v modelu nezastupitelnou pozici. Například zařízení je v organizační vrstvě situováno v *místnosti* a ve fyzické vrstvě je to samostatné zařízení.

Organizační struktura



Fyzicko-logická struktura





Úkolem organizační vrstvy je celkově zachytit a podchytit veškeré vazby, které specifikují metainformace o lokaci zařízení v rámci struktury organizace.

Nejvyšší vrstvou organizační struktury je **Organizace**.

V našem případě se jedná o Západočeskou univerzitu. Organizace má seznam odkazů na jednotlivé lokality.

Lokalita je označení pro celistvou, jednolitou a geograficky oddělenou oblast, ve které se nacházejí jednotlivé budovy, případně jednotlivé objekty složené z budov.

Budova je obalující prvek, který popisuje soubor místností a jednotlivých zásuvek. Případně může podchycovat i jednotlivá místa, kde se nachází jednotlivá zařízení.

3.1.2 Analýza prováděných operací

Práce síťového administrátora zahrnuje celou řadu úkolů. Mezi hlavní patří správa a nastavování VLAN po celé trase (od počátečního bodu - přes veškeré switche po trase do koncového bodu). Tato virtuální podsíť (Virtual Local Area Network) se používá pro komunikaci u jednotlivých oddělení - vedení, správci, ... Při práci s VLAN je nutné zahrnout následující operace:

- Přidávání
- Odebírání VLAN
- Parametrizace VLAN

Administrátoři jistě využijí další operace, které by jim umožnili zjednodušit správu síťových prvků. Mezi operace, které každý správce vykonává je řazena i správa uživatelských účtů na jednotlivých síťových prvcích.

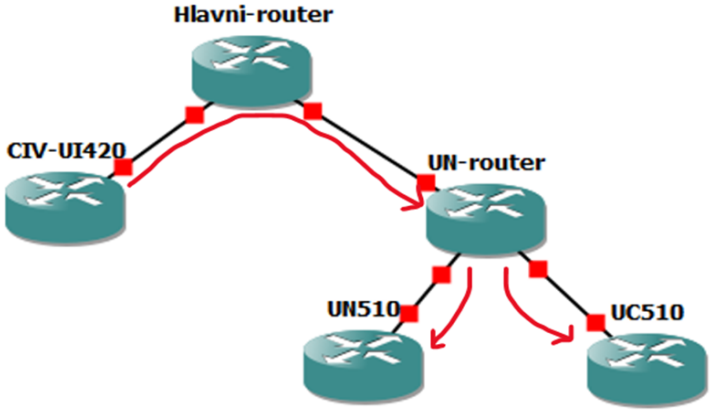
Přidávání uživatelských účtů na všechny síťové prvky s sebou nese jistou zátěž, kterou by mohlo být možné pomocí této práce eliminovat.

3.1.3 Analýza vstupních dat

Základem jsou soubory, které poskytují cenné informace o souhrnném stavu sítě. Tyto informace mohou být uchovávány v souborech, které mají různý formát. Důležité je, aby formát umožňoval hierarchické ukládání dat do struktur. Pro tento účel můžeme použít například **CSV** nebo **JSON**. V našem případě jsou používány Cisco CFG soubory.

Ukázka konfiguračního souboru Cisco je uvedena v příloze.

Dále bude uveden vstupní ukázkový soubor konfigurace VLAN



Obrázek 3.1: Ukázka přenosu VLAN přes síťovou infrastrukturu

Ukázka kódu 3.1: Ukázkový konfigurační soubor CISCO - VLAN

```
VLAN     Interface     IP-Address       Status      Protocol
2         Fa0/0                      192.168.1.1      up
    ↪          up
3         Fa0/1                      10.0.0.1         down
    ↪        down
4         Serial0/0                  172.16.1.1       up
    ↪          up
```

3.2 Návrh řešení

Při navrhování celkového řešení bylo důležité vycházet z faktu, že se síť chová jako neorientovaný graf. Z toho důvodu byly veškeré struktury (použité pro ukládání struktur do databází) abstrakcí grafu na úrovni SŘBD (Systému řízení báze dat). V rámci ukládaných struktur bylo nutné řešit veškeré souvislosti napříč výše zmíněnými vrstvami.

Z celkového pohledu je důležité problém posoudit z několika následujících perspektiv:

- Požadavky na nástroje
- Výběr nástrojů
- Přístup k propagování změn
- Rozdělení vrstev - organizační ...
- Přístup k ukládání source-of-truth

- Realizace pipeline v prostředí CI/CD

Každá z těchto vlastností je důležitá pro zamezení vícenásobné změny a pro ochranu před **DRIFTEM** - rozdílným pojetím současného stavu na cílové infrastruktuře a stavu, který nástroj rozeznává jako aktuální na daných nodech.

3.2.1 Požadavky na technologie

Při správě síťové infrastruktury je nutné dodržovat základní návyky a standardy. Nejdůležitější parametry pro nástroje spravující síťovou infrastrukturu jsou:

- Bezagentový přístup
- Idempotentnost
- Modularita
- Deklarativní i imperativní přístup k problematice

Podíváme-li se na celý cyklus správy síťové infrastruktury od začátku, je možné najít v přípravných fázích situace, kdy je nutné do mnoha nenakonfigurovaných zařízení vložit předdefinovanou množinu operací, aby bylo možné následně switche shlukovat do větších celků a ty centrálně spravovat. Z tohoto důvodu je nutné, aby nástroj umožňoval správci bezagentový přístup k technologii/zařízení.

Asi nejdůležitější vlastností pro nástroj je idempotentnost. Tento termín popisuje způsob přístupu k aplikování jednotlivých operací. Změny v konfiguraci jsou aplikovány, je-li cílový stav odlišný od námi definovaného stavu (resp. stavu, který je nyní na zařízení). Výhoda u idempotentnosti je v přístupu, kdy aplikace změn je prováděna pouze v potřebných případech. Z toho důvodu je ušetřeno mnoho zdrojů a nejsou aplikovány a přepisovány změny několikrát po sobě.

3.2.2 Návrh technologií pro realizaci

Pro potřeby automatizace správy a nasazování infrastruktury je využíváno několik druhů nástrojů. Mezi hlavní zástupce můžeme zařadit Ansible a Terraform.

Při srovnání technologií Terraform a Ansible zjistíme odlišné přístupy k infrastruktuře jako takové. Terraform používá deklarativní jazyk jako přístup ke konfiguraci. Konkrétně se jedná o Hashicorp Configuration Language, který popisuje požadovaný stav infrastruktury. Stejně jako Ansible

umožňuje modularitu a tvorbu proměnných. Terraform je zaměřen spíše na poskytovatele cloudových řešení. Terraform využívá lokálního state managementu. Informace o stavu infrastruktury je udržována v lokálním souboru s názvem state file (přípona .tfstate). Tento soubor obsahuje kompletní aktuální stav nasazených zdrojů (obrazů/image/VM) a používá se v Terraformu k udržování konzistence a pro sledování změn v infrastruktuře, které se mají provést.

Pro srovnání: v Ansible je možné použít k definici deklarativní konfiguraci infrastruktury pomocí YAML souborů a dále také umožňuje k problematice imperativní přístup pomocí tzv. playbooků - souborů, které popisují jednotlivé kroky k dosažení požadovaného stavu. Ansible je jeden z agentless nástrojů, tedy není potřebné instalovat jakoukoliv technologii na cílové zařízení (jen mít dostupné SSH). Hlavní a možná nejdůležitější vlastnosti Ansible je **idempotentnost**. Idempotentnost zajišťuje provádění jen těch změn, které jsou nutné pro přeměnu cílového node na námi definovaný stav. Neméně důležitá je i modularita, která umožní pomocí rolí přesně definovat množinu aplikovaných kroků k dosažení cílového stavu.

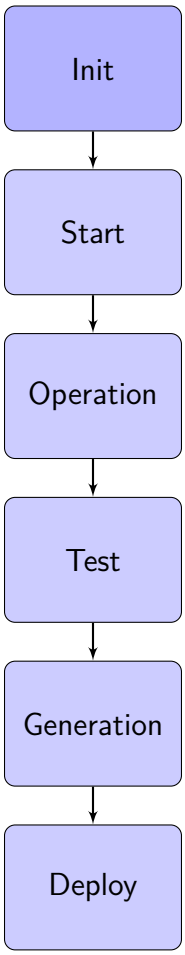
Z hlediska definovaných požadavků na nástroj pro automatizaci síťové infrastruktury je v tomto případě nejvhodnější nástroj Ansible.

3.2.3 Popis procesu nasazení

V rámci diplomové práce je průběh celého procesu rozdělen do následujících 5 fází:

- Init - Příprava
- Start - ETL
- Operation - Provádění
- Test - Otestování
- Generate inventory / group_vars - Generování
- Deploy - Provádění změn

Celý proces zahrnuje přípravné fáze pro nahrávání dat. Dále se přesouvá k extrakci a nahrání samotných dat. Na nich jsou prováděny naše operace, které jsou otestovány. Pro celou infrastrukturu, která se bude nasazovat, je vygenerován seznam informaci, které jsou pro tuto operaci potřeba. Na závěr je vše ověřeno a spuštěno.



Obrázek 3.2: Diagram of data flow

Na začátku celého procesu je část **Init**, která má na starost přípravu infrastruktury a nastavení všech schémat s tím spojených. Tato část se většinou provádí jen jednou za celý cyklus softwaru. Při změně na používaném schématu je nutné odděleně vést informace o těchto změnách (migracích). Migrace se musí vždy provést při zakládání nového environmentu. Tato práce tvoří základ celého systému, proto na konci této diplomové práce není připravená žádná migrace.

Další v pořadí je operace **Start**. Tato část je také důležitá pro přípravu a nahrávání dat. V této části se vstupní soubory (v našem případě konfigurační soubory Cisco ve formátu .CFG a .VLAN výstupy) zpracují a převedou do YANG modelu. Obecně bylo využito technologie ETL (Extraction - Transformation - Load). Každá z těchto částí je důležitá pro tvorbu celkového source-of-truth.

Celou ETL pipeline můžeme popsat takto:

- **Extrakce** - v rámci této fáze se ze vstupních souborů převádí vstupní konfigurační soubory do pole, které je následně předáváno do další fáze
- **Transformace** - transformace se postará o převod vstupního pole konfigurací do podoby YANG objektu/modelu a příslušné JSON struktury a ta, stejně jako v předešlém kroku, vstupuje do následující části
- **Load** - poslední část celého procesu, která se postará o nahrání schémat v příslušném formátu do databáze (NoSQL)

Níže je uvedena (v příloze) podoba výstupní struktury dat používaná v systému jako source-of-truth. Využívá se jako zdroj důležitých a platných dat pro zápis konfiguračních parametrů. Data zohledňují veškeré parametry, které byly potřebné pro provádění a uchovávání.

4 Realizační část

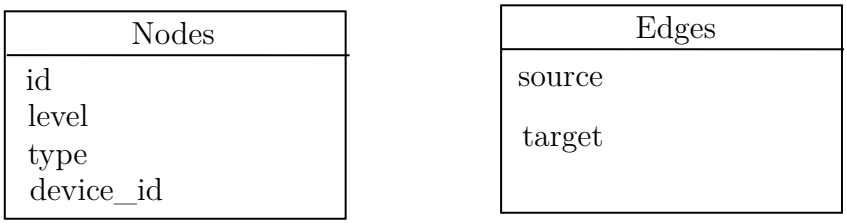
V této kapitole bude výstupní nástroj této diplomové práce popsán z pohledu zadávání operací - konkrétní specifikace vstupních dat, dále z bližšího pohledu na programovou realizaci prováděných operací a popisu všech dostupných a k nim příslušných parametrů a operací.

Diplomová práce si klade za cíl zjednodušovat a standardizovat složitější operace, které se provádí v návaznosti na situační povědomí o jednotlivých prvcích a jejich konfiguracích.

4.1 Design systému

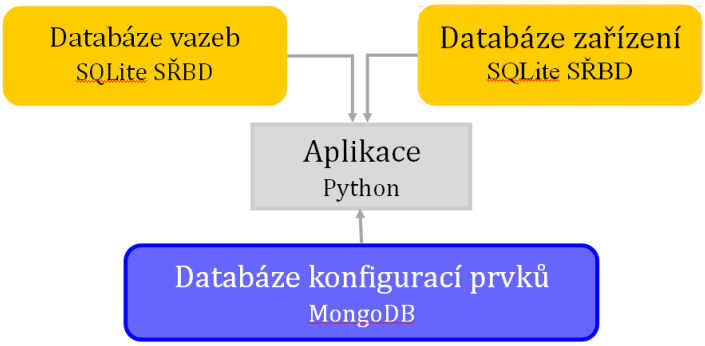
Vazby mezi jednotlivými síťovými prvky jsou reprezentovány grafovou strukturou. Při podchycení vazeb je nutné mít struktury potřebné pro ukládání grafu. Graf je možné realizovat několika způsoby. Nejčastější realizace je pomocí vrcholů a hran. V rámci Pythonu je vazba mezi databází v SQLite formátu realizována pomocí Flask ORM technologie, která zajišťuje synchronizaci jednotlivých datových objektů s jejich příslušnými obrazy v databázi. Jednotlivé objekty jsou následně přetvořeny do příslušných struktur, které tvoří graf počítačové sítě. Počítačová síť je realizována jen pomocí jednotlivých síťových prvků jako vrcholů (nodes) a propojů - hran (edges). Nad tímto grafem jsou následně aplikovány jednotlivé operace pro distribuci VLAN apod.

V rámci tvorby grafu je nutné vytvořit následující schéma:



Obrázek 4.1: ERA diagram grafu

Systém při práci komunikuje s několika databázemi, které uchovávají potřebné informace o síti jako takové. Při komunikaci jsou veškeré informace



Obrázek 4.2: Schéma komunikace systému

následně překlápěny pomocí ORM do objektů, které jsou následně využívány v jednotlivých implementacích.

Pro potřeby ukládání databáze vazeb byla použita jednoduchá relační databáze pro jednoduchost integrace s technologií ORM. Tato vlastnost byla důležitá pro výběr této technologie. Dále bylo stanoveno minimalistické množství hodnot, které bude navázáno na graf. Díky tomu nebylo nutné využít funkcí grafové databáze. SQlite je souborová databáze, která umožňuje uchování veškerých informací a neomezuje programátora při implementaci těchto algoritmů. Tato databáze primárně slouží pro podchycení vazeb při transportu VLAN skrz síť.

Pro potřeby aplikace bylo vhodné použít jazyka Python díky široké dostupnosti jednotlivých knihoven pro práci s grafy, napojitelnost na databázi a práci s YANG modely. Python pro tvorbu aplikace byl vhodný vzhledem k předchozím zkušenostem autora. Aplikace je také multidatabázová. Napojitelnost na jednotlivé typy databází - SQLite a MongoDB - byla relativně náročná na realizaci. V Pythonu byla řada knihoven, které tento náročný proces zjednodušily a umožnily tvorbu konceptů, které následně byly aplikovány při ukládání YANG modelů pro definici konfigurací jednotlivých síťových prvků.

Pro ukládání jednotlivých konfigurací síťových prvků byla vybrána databáze MongoDB z důvodu předchozích zkušeností programátora s touto databází. Databáze umožňuje v cloudové variantě navíc implementaci Triggerů, které mohou být využity například při změně v databázi. Databáze konfigurací uchovává následující informace:

- `_id`: Id daného zařízení, které umožní párovat jednotlivá zařízení napříč databázemi
- `hostname`: Hostname daného zařízení, pomocí kterého jsou párovány

jednotlivé konfigurační soubory

- `vlan`: Seznam nasazených jednotlivých VLAN
- `users`: Uživatelské účty, které mají být deployovány do switche
- `ietf-interfaces:interfaces`: Objekt, který uchovává kompletní YANG model, který uchovává informace o jednotlivých interfacech

Díky podpoře triggerů v databázi byl realizován i trigger, který podchytí změny provedené při mazání v databázi. Tyto změny musí být také aplikovány následně na switchích. Tyto změny jsou propisovány do speciální tabulky backup, která uchovává informace o změnách. Ukázka triggeru v MongoDB je uvedena v příloze.

Trigger vezme informace o portu, který byl smazán, a uloží je do tabulky backup.

Databáze zařízení uchovává informace o jednotlivých switchích a routerech. Tato zařízení mají jednotlivé porty a propoje, které musí být podchyceny. V databázi je také podchycena vazba organizace na jednotlivé budovy, budovy na místnosti a místnosti na zařízení. Zařízení má k sobě dodefinované jednotlivé porty se svými parametry.

4.2 Zadávání operací

Tato diplomová práce je zaměřená na jednotlivé operace spojené s úpravou interfaců a správou VLAN na síti. Parametry pro provádění operací jsou zaneseny do vstupních souborů, které popisují požadovaný cílový stav.

4.2.1 Popis zadávání

Hlavním bodem celé aplikace je soubor `input.json`. V něm se definují potřebné parametry spojené s prováděnou operací.

Soubor se vstupními parametry pro běh aplikace je uveden v příloze. Parametry konfiguračního souboru popisují následující informace:

- data: Obsahuje informace o datových úložištích používaných v procesu.
- datastore: Název datového úložiště, ve kterém jsou uchovávána data ve formě grafu.
- datasource: Název datového zdroje nebo databáze uloženého v souboru.

- global: Obsahuje globální informace o provozním prostředí.
- scope: Určuje rozsah nebo prostředí, ve kterém se provádějí operace. V tomto případě je označeno jako "staging".
- provider: Definuje poskytovatele služby nebo systému, který je používán v procesu.
- endpoint: Adresa nebo URL koncového bodu, ke kterému se připojujeme pro provádění operací.
- username: Uživatelské jméno pro ověření u poskytovatele služby.
- password: Heslo pro ověření u poskytovatele služby.
- operation: Určuje typ operace, která se má provést.
- type: Typ operace, zde specifikovaný jako "add"(přidání).
- subject: Specifikuje, na jakém objektu se má operace provést. Zde je uvedeno, že se jedná o operaci na VLAN.
- parameters: Obsahuje parametry nebo hodnoty, které jsou potřebné pro provádění operace.
- vlan_id: Identifikátor VLAN, který se má přidat. Zde je uveden jako "30".
- vlany_name: Název VLAN, který se má přidat. Zde je uvedeno jako "Test VLAN".
- src_device: Zdrojové zařízení, se kterým se má provádět operace. Zde je uvedeno jako "CIV".
- src_port: Zdrojový port na zdrojovém zařízení.
- dst_device: Cílové zařízení, na kterém se má provádět operace. Zde je uvedeno jako "UN-A".
- dst_port: Cílový port na cílovém zařízení.

Data reprezentují informace o zdrojích dat, datových úložištích a výstupních souborech používaných v procesech správy a údržby dat. Datastore je název vstupního datového úložiště (v našem případě databáze, která je reprezentována 1 souborem), ve kterém je uchováván v relační formě graf dané sítě. Datasource označuje název datového zdroje či databáze.

Globální sekce obsahuje informace o daném prvku, bližší specifikaci jeho provozního nastavení a důležité informace pro aplikaci změn. Scope určuje rozsah nebo prostředí, ve kterém se provádějí operace. V tomto případě je označeno jako "staging".

Provider sekce ukazuje navázání na poskytovatele obecné další služby nebo na jiné subsystémy použitelné při úpravách a specifikaci dalších parametrů, které by byly použity v procesu. Endpoint je vzorová adresa koncového bodu, ze kterého by byly brány informace pro provádění námi definovaných operací. Username je uživatelské jméno pro autentizaci do výše definované závislosti - systému - který bude poskytovat služby / data, a password je příslušné heslo pro ověření přístupu k němu. Sekce je zajímavá ukázkou využití vazby definice proměnných uložených v separátním souboru `vars.json`, které jsou donačítány do tohoto systému pro potřeby externích souborových zdrojů pro získávání dat.

Operation určuje typ operace, která bude prováděna a má být blíže parametrizována a specifikována. Type označuje typ operace, zde specifikovaný jako "add"(přidání) nebo třeba "delete". Subject specifikuje, na jakém objektu se má operace provést. Zde je uvedeno, že se jedná o operaci na VLAN.

Parameters obsahuje parametry nebo hodnoty, které jsou potřebné pro provádění operace. Vlan_id je identifikátor VLAN, který se má přidat, zde uveden jako "30". Vlan_name je název VLAN, který se má přidat, zde uvedeno jako "Test VLAN". Src_device označuje zdrojové zařízení, se kterým se má provádět operace, zde uvedeno jako "CIV". Src_port je zdrojový port na zdrojovém zařízení.

Dst_device je cílové zařízení, na kterém se má provádět operace, zde uvedeno jako "UN-A". Dst_port je cílový port na cílovém zařízení.

Ukázka kódu 4.1: Externí soubor proměnných použitý jako zdroj bázových informací

```
{
  # Ukazka definice promennych pro input.json
  "endpoint": "Endpoint",
  "username": "User",
  "password": "Password",
  "vlan_template": "vlan_template.j2"
}
```

V rámci ukázky kódu 4.1 je možné vidět definice jednotlivých proměnných pro soubor input.json. Následující proměnné jsou následně používány jako rozšíření existujících schopností systému načítat data jen z 1 centralizova-

ného systému vstupních systémových parametrů

4.3 Bližší popis operací

V rámci systému byla implementována řada operací. Tyto operace mají za cíl usnadnit a zpřehlednit správu síťových rozhraní se zaměřením na interface a VLAN. Na tomto základě bylo vytvořeno několik scénářů, které mají za úkol provádět předem definované operace pro například transport VLAN nebo pro úpravu parametrů na jednotlivých interfacech.

4.3.1 Práce nad interfacem

Změny parametrů (nad specifickým interfacem) se provádí v nástroji Ansible pomocí LOOP smyčky, která prochází přes všechny záznamy o rozhraních uvedených v group_vars. Příslušné hodnoty se následně aplikují na daném místě jen pokud se jedná o změnu. V opačném případě není aplikována žádná z konfiguračních změn.

Parametry, které je možné nastavit u každého interface:

- `enabled` - určuje, zda je interface aktivní
- `admin-status` - indikuje, v jakém stavu (například vzhledem k protokolu RIP) se daný interface nachází
- `description` - jméno interface - Stará se o manipulace při pojmenovávání daného interface. Je zde možné podchytit i virtuálně jiné vazby či komentáře nad tímto portem. Důležité to může být pro určování fyzické topologie sítě z konfigurace. Díky identifikaci protistrany (hostname daného zařízení) - většinou jako komentář u daného portu - můžeme určit propojení na další prvky. Tato vlastnost pak umožňuje lépe identifikovat veškerá zařízení připojená k jednotlivým portům a sjednotit a centralizovat správu celé infrastruktury.
- `type` - definuje, o jaký typ interface se jedná. Bude-li potřebné změnit typ na jiný druh, je možné s tímto parametrem pracovat
- `link-up-down-trap-enable` - se stará o trapování (zachytáváním) události na daném interface pro potřeby debugování a hledání chyb a závad. Nebudou-li tyto informace potřebné, je možné tento senzor vypnout.
- `statistics` - statistiky o daném portu - úprava například poslední manipulace skriptem s tímto portem

- `ietf-ip:ipv4` - nejzajímavější ze všech parametrů - je určen pro nastavování IPv4 (pro IPv6 je speciální kategorie podobná této). Je možné v rámci pole address vložit více adres na 1 interface. Dále je zde možné nastavovat parametry síťové masky (netmask), a to ve formátu například: 255.255.255.0 (mask /24) a také origin = static, tj. neměnnost a statičnost přidělených IP adres na daný interface

- `name` - popisuje referenci na fyzický interface na switchi. Editace tohoto pole není třeba

4.3.2 Manipulace s VLAN

Nejdůležitější částí této diplomové práce je manipulace s VLAN po sítí.

Při manipulaci s VLAN je potřebné podchytit následující situace:

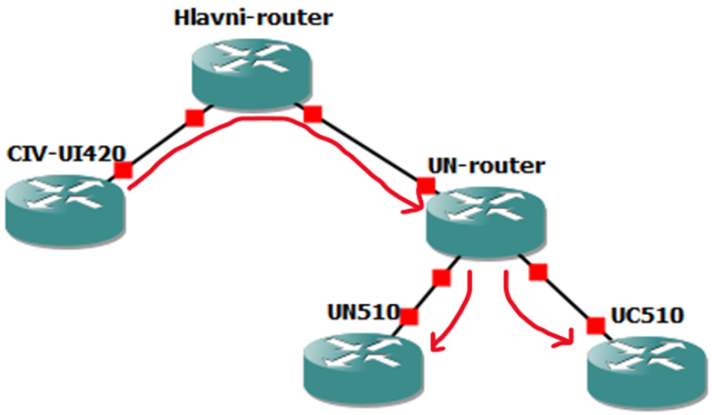
- Změna VLAN name
- Přidání VLAN na jediném switchi
- Distribuce VLAN ze switche A do switche B
- Odebrání VLAN ze switche A do switche B
- Distribuce v rámci logické jednotky

Při změně VLAN name je nutné projít veškerá zařízení, pro která jsou definovány změny. Následně jsou změny zavedeny do source-of-truth. Tyto změny jsou následně pomocí proměnných v Ansible vypropagovány do všech zařízení. Důležité při tomto úkonu je definovat veškerá zařízení, kterých se tento úkon týká. Nadále je potřebné uvést i interface, který je navázán na dané číslo VLAN.

Při přidávání VLAN na jednom switchi se v rámci YANG modelu změní parametr u příslušné VLAN. Změna se následně propíše stejně jako u předchozího řešení.

Distribuce VLAN ze switche A do switche B není jednoduchá úloha. V rámci distribuce mezi dvěma zařízeními musí být zkontrolována a nalezena řada informací. Nejprve je nutné zjistit, kterých prvků se přidávání VLAN týká - resp. zda jsou na **fyzické** trase ze switche A do switche B. V tomto případě je nutné zapojit fyzickou vrstvu a zjistit propojení jednotlivých zařízení. Dále je nutné zapojit i fyzicko-logickou strukturu, podle které se určuje, na který port se daná změna musí aplikovat. Jinak by například mohlo dojít k neoprávněnému přístupu do Management VLAN, která slouží pro přístup k administraci jednotlivých zařízení. Jedná se tedy o port

UP (uplink - směrem k hlavnímu routeru) a DWN (downlink - směrem k dalším zařízením). Nadále je také nutné změnu aplikovat na VTP serverech tak, aby se změna propagovala na všechna potřebná zařízení. Akce je následně aplikována - opět jako v předchozím případě - do Source-of-truth a je následně pomocí Ansible propsána do jednotlivých zařízení.



Obrázek 4.3: Schéma prováděné operace

Při distribuci VLAN v rámci logické jednotky (budova, lokalita ...) je důležité vycházet z vazeb a logických pro-vazeb. Při distribuci se prochází graf z jeho vrcholu (například hlavní router pro budovu) a prochází skrz veškeré podřízené struktury - objekty (pro budovu - místnosti) a přes fyzicko-logickou strukturu se zapíší a propíší jednotlivé úkony do podřízených zařízení. V tomto případě se prochází abstraktní struktura strom (tree), která je uložená v databázi. Průchod je realizován pomocí algoritmu BFS (Breadth-First Search), aby bylo možné vložit a aplikovat VLAN ve všech podřízených objektem (místnostech). Celá operace je zanesena na schématu 4.3.

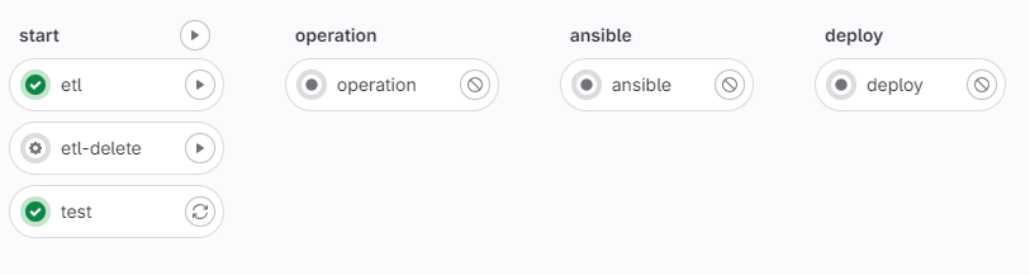
4.3.3 Distribuce nových uživatelů na infrastrukturní prvky

Proces nasazení admin uživatelů na switchi nebyl vůbec náročný. U definice jednotlivých uživatelů je jen seznam uživatelů, kteří mají být vytvořeni v cílové destinaci. Pro omezení skupiny zařízení (kam mají být definovaní uživatelé nasazeni) je nutné změnit parametr hosts v playbooku.

Následně je v příslušném YAMLu nutné nadefinovat pro pole **users** jednotlivé záznamy pojmenované jako **username** a **password**.

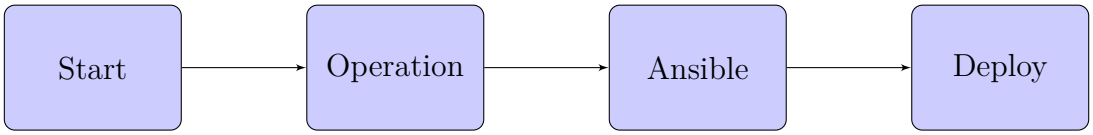
4.4 CI/CD pipeline

Pipeline slouží pro automatizovanou správu spouštění a triggerování jednotlivých operací. Operace jsou zde rozděleny do jednotlivých stage. Každá stage je spouštěna dle stanoveného pořadí a dle stanového pravidla. CI/CD pipeline pro diplomovou práci byla realizována v Gitlab CI/CD na CIV.



Obrázek 4.4: Gitlab pipeline, zdroj: ipmil.civ.zcu.cz

Pipeline je možné rozdělit do několika fází. Tyto fáze znázorňují jednotlivé operace, které se vytváří nad sítí. Bližší schéma operací je znázorněno na obrázku 4.4



Obrázek 4.5: Soubor operací v CI/CD pipeline

4.5 Pipeline pro správu a nasazení počítačových sítí

Tento rozsáhlý pipeline představuje komplexní proces správy a nasazování počítačových sítí, který zahrnuje různé fáze, úkoly a strategie pro efektivní automatizaci a ověření. Každá část tohoto pipeline má svůj specifický účel a přispívá k celkové správě a provozu sítě.

4.5.1 Proměnné a jejich využití

Proměnné definované na začátku pipeline poskytují možnost konfigurace a nastavení různých parametrů, jako je například cílové prostředí, do kterého

se má nasazovat. Tím umožňují flexibilitu a snadnou úpravu chování pipeline podle potřeb uživatele.

4.5.2 Fáze pipeline

Jednotlivé fáze znázorňují jednu sadu operací, která je prováděna v rámci celého cyklu nasazování nové verze konfigurace sítě.

Fáze "start"

Tato fáze zahrnuje inicializační úkoly, které jsou nezbytné pro spuštění dalších operací. Úkoly `etl` a `etl-delete` jsou zodpovědné za inicializaci a případné odstranění zdrojů, které již nejsou potřebné pro další kroky.

- **Úkol `etl`**: Spouští skript pro inicializaci zdrojů.
- **Úkol `etl-delete`**: Slouží k manuálnímu odstranění zdrojů. Jeho spuštění je řízeno pravidlem `manual`, což znamená, že vyžaduje ruční zásah uživatele.

Fáze "operation"

Tato fáze je prázdná, což naznačuje, že zatím nebyly definovány žádné úkoly. Může sloužit jako místo pro přidání dalších operací nebo rozšíření pipeline v budoucnosti.

Fáze "ansible"

V této fázi se provádí generování souborů pro nástroj Ansible, který je použit pro konfiguraci sítě a další operace. Úkol `generate_ansible_files` je odpovědný za tuto činnost.

Fáze "deploy"

Poslední fáze pipeline je zodpovědná za samotné nasazení sítě. Nejprve jsou provedeny přípravné kroky, jako je inicializace SSH agenta a nastavení prostředí. Poté je spuštěn úkol `deploy-network`, který provádí nasazení sítě pomocí Ansible.

4.5.3 Pravidla spouštění úkolů

Některé úkoly mají specifická pravidla pro jejich spouštění, což přispívá k flexibilitě a řízení procesu nasazení sítě.

- Úkol `etl-delete` je spouštěn pouze manuálně, což umožňuje uživatelům ručně ověřit a potvrdit operace s potenciálně významnými dopady.
- Úkol `deploy-network` je spouštěn buď automaticky, pokud je pipeline spuštěna jinou pipeline, nebo manuálně, pokud je spuštěna ručně.

Celkově tento pipeline poskytuje strukturovaný a automatizovaný proces pro správu a nasazení počítačových sítí zahrnující inicializaci, testování a nasazení pomocí nástroje Ansible. Jeho modulární a flexibilní povaha umožňuje snadné rozšíření a úpravy podle potřeb uživatele.

Pro potřeby CI/CD byla vybrána platforma Gitlab, protože je dostupná na výpočetním prostředí CIV.

Ukázka definice Gitlab pipeline je uvedena v příloze.

5 Testování

V rámci testování počítačových sítí bylo užito jednotkových testů nebo-li Unit testů. Unit testy byly základem pro ověřování a otestování jednotlivých funkčností u naprogramovaných automatizačních skriptů.

Pro ověření správného fungování a správné integrace napříč dostupnými moduly aplikace byly vytvořeny integrační testy prověřující soudržnost aplikace jako celku. Každá část aplikace je pro potřeby simulace komunikace mezi jednotlivými částmi aplikace a pro nalezení případných komunikačních bariér nasimulována pomocí jednoduchých mocků.

Bez dostatečného otestování aplikace by automatizace nasazování počítačových sítí byla nerealistická, případně by vykazovala známky chybového chování. Testování je také součástí integrační pipeline v rámci CI/CD. Testování zde neprobíhá jen na úrovni jednotkových testů, ale i na úrovni správnosti vstupních parametrů, kontroly dostatečného množství vstupních parametrů pro provádění jednotlivých operací, testů přístupnosti k zařízením a na závěr jednoduchých testů (správně provedeného nasazení) na zařízení. Nicméně, vzhledem k složitosti a návaznostem v současných sítích, je důležité důkladné testování těchto automatizovaných procesů. V případě chybného fungování by byly veškeré výhody automatizace (zjednodušení a zefektivnění procesů) nevalidní. Chyby v aplikaci by neminimalizovalo riziko chyb a zvýšil by se počet neplánovaných výpadků sítě.

5.1 Problémy a výzvy při hledání návrhu ověřování automatizovaných procesů nasazení sítí

Testování pro potřeby zjednodušení, automatizace procesů, nasazení a správy sítí přináší oproti testování webových aplikací různé komplikace a liší se od standardního testování. Přináší s sebou i vyšší složitost a některé výzvy. Problematické části jsou zaneseny v následujících bodech:

- **Heterogenita prostředí**: Sítě se často skládají z prvků několika providerů. Každý z těchto prvků má různé konfigurační parametry. Navíc se stejná konfigurace může lišit napříč verzemi operačního systému jednotlivých switchů a routerů. Dále významnou roli v heterogenitě hraje topologie. Jiná konfigurace bude platit na leaf node a jiná konfigurace

pro spine node. Navíc většina leaf node a spine node má často stejný typ operačního systému. Testování musí být schopno simulovat různé typy prostředí a námi definované topologie, aby se zajistilo, že automatizované procesy je možno použít na různé druhy síťových architektur a organizací síťových prvků do celku. Dále je potřeba, aby i v těchto případech bylo možné ověřit, zda prvky fungují správně ve všech definovaných scénářích pro dané prostředí a typy vazeb v síti. ¨Důležité je podchycení jednotlivých vazeb a návazností na sebe.

- **Správnost konfigurace**: Automatizované nástroje mohou dle našeho nastavení vytvářet předem stanovené konfigurace sítě, které jsou automaticky testovány vůči definicím a stavům na základě definovaných pravidel a templates (šablon). Šablony je nutné strukturovat pro jednotlivé typy operačních systémů, dále také dle architektury procesoru (mipsbe, x86, ARM), verze operačního systému, typu síťového prvku (leaf, spine), ale také dle standardů při konfiguraci v dané organizaci. Je důležité, aby se nezapomnělo na výjimky, které se vyskytují v každé větší síti. I tyto odchylky musí být schopen systém podchytit pomocí jednotlivých funkcí a otestovat, že není narušeno jejich fungování. Je nezbytné ověřovat pomocí testů, zda tyto předpřipravené generované konfigurace odpovídají očekávaným výstupům a jsou správně aplikovány na příslušná síťová zařízení a definovaná síťová rozhraní - interfaces.

- **Bezpečnost nasazení konfigurace**: Nasazení automatizovaných procesů musí být v souladu s bezpečnostními a konfiguračními standardy. Následně může být otestováno například pomocí Open Policy Agenta. Výsledek testování musí být vždy kladný. Součástí této diplomové práce je i nástroj na kontrolu základních bezpečnostních nastavení na jednotlivých prvcích. Testování a ověřování je nutné provádět pravidelně a ideálně z hlediska bezpečnosti i po každé provedené změně, aby se zabránilo možným zranitelnostem, průnikům do systému a bezpečnostním hrozbám v síťovém prostředí. Integrace do procesu CI/CD ulehčuje bezpečnostním technikům práci a s nástrojem této diplomové práce je možné odchytit základní chyby při bezpečnostní konfiguraci prvků.

- **Výkon a škálovatelnost**: V rámci testování musí být zahrnuty i jednotlivé metriky pro hodnocení výkonu automatizovaných procesů a jejich výtěžnosti na testovací síti pro potřeby pečlivého plánování testů a škálování jednotlivých komponent pro zvyšování výkonu. Je důležité

počítat při testování s nárůstem výkonových požadavků na infrastrukturu a analyzování délky běhu skriptů. Část skriptů diplomové práce je možné zrychlovat (část psaná v pythonu), ale zbylá část (převážně Ansible) je závislá na schopnostech a optimalizaci jednotlivých komponent systémů, případně dle konfigurace vstupních parametrů a prováděných testů.

- **Integrace a návaznost s existujícími systémy a síťovými prvky**: V rámci automatizovaných procesů je nutné před spuštěním skriptů zkontrolovat, zda je možné se k prvkům vůbec připojit. Tato část je realizována interně v rámci Ansible. Následné testy mohou být spuštěny právě tehdy, jsou-li všechny nody dostupné. Toho lze docílit virtualizací jednotlivých komponent sítě. Takto je to řešeno například v rámci této diplomové práce. Automatizace a integrace je následně prováděna na různých vrstvách. Pro komunikaci se source-of-truth jsou vytvořeny testovací případy už od vývojářů systému, které procházejí bez chyb. Následné testy a vazby na SSH příkazy a spojení s prvky probíhá na úrovni nejnižších vrstev Ansible. Tedy není nutné, aby bylo testováno z pohledu uživatele. Testování a integrace probíhá jen na úrovni systému doprogramovaného nad již hotovými komponentami a pro potřeby testování soudržnosti celkového řešení a podchytávání návazností na jednotlivé připravené moduly, a to moduly pro práci s jednotlivými návaznostmi na jednotlivé vrstvy organizační, fyzické či logické vrstvy nebo vazby mezi komponentami na úrovni návrhu software. Komponenty už z principu musí spolu komunikovat. Tedy testování je nutné provádět buď pomocí simulace nebo pomocí mockování jednotlivých entit a tříd. Je nutné také vytvořit koncept pro testování organizačních struktur, který není možné podchytit pomocí unit testů a integračních testů. Nejnižší úroveň těchto testů není proto dostačující. Automatizované procesy musí být testovány i z hlediska následné interoperability a přípravy integrace s existujícími a následujícími systémy a pro procesy v organizaci.

5.2 Metody testování automatizace nasazení sítí

K testování automatizace nasazování počítačových sítí lze použít různé metody a postupy. Některé z nejčastěji používaných metod zahrnují:

- **Manuální testování**: Manuální testování zahrnuje ruční ověřování

funkčnosti a správnosti automatizovaných procesů nasazení sítí. Tato metoda je časově náročná a náchylná k chybám, ale může být užitečná pro ověření komplexních scénářů a chování sítě. Tato metoda byla nejvíce využívána při testování této práce.

- **Automatizované testování**: Automatizované testování využívá skriptování a automatizaci k provádění opakovaných testů automatizovaných procesů nasazení sítí. Tato metoda umožňuje opakované a konzistentní testování bez manuálního zásahu.

5.3 Závěr testování

Testování automatizace nasazování počítačových sítí je klíčovým prvkem v procesu implementace a správy sítí v moderním IT prostředí. Správně provedené testování může pomoci minimalizovat rizika chyb a neplánovaných výpadků sítě a zajišťuje spolehlivost nasazení a výkon automatizovaných procesů nasazení sítí. Použité testovací scénáře proběhly všechny správně.

6 Zhodnocení výsledků

V rámci této diplomové práce bylo realizováno více druhů operací a příprava pro další rozšíření, byla vytvořena sada skriptů pro testování spolehlivosti nasazení sítě, automatizaci přípravy dat, konverze, nahrávání, příprava struktur pro uchovávání vazeb, analyzátor vazeb mezi jednotlivými komponenty a rozšíření templatovací engine TTP o další šablony. Výsledkem je zcela funkční aplikace otestovaná na virtualizovaném testovacím prostředí pomocí GNS3. Aplikace byla následně otestována i s výstupy testů. Veškeré testy byly zcela funkční. Nasazování probíhalo na drobnější testovací síti. Byly prováděny operace od nejzákladnějších změn popisků dat až po celkové nasazení VLAN ze serverovny do budov.

Celý proces byl založen na nastudovaných metodách správy sítě a její automatizace. Hlavním zdrojem pro práci se sítí byla kniha Johna Capobianca *Automate your network* [5]. Bylo vyzkoušeno několik odlišných metod a byly porovnány a otestovány jejich výsledky.

6.1 Typy realizovaných experimentů

V rámci diplomové práce byly realizovány skripty pro podchycení vazeb u virtualizované sítě na bázi GNS3. Konfigurační soubory této sítě byly vloženy do příslušných složek a následně spuštěny veškeré operace, které slouží pro nahrávání a správu operací nad sítí. Tato data byla propsána do databáze znalostí o sítí (source-of-truth). Následně byly doimplementovány moduly, které umožňují správu uživatelských účtů na jednotlivých zařízeních.

V rámci diplomové práce na téma *Automatizace nasazení a správy síťových zařízení* byla provedena analýza, v níž byly představeny současné a moderní pohledy na automatizaci a správu sítě malé i rozsáhlejší (univerzita či korporátní prostředí). Diplomová práce poskytuje přehled technologií a poskytuje nástroje pro nejdůležitější oblasti práce. Cíle práce jsou rozvedeny v následující kapitole o dosažených výsledcích.

6.2 Realizované činnosti

Diplomová práce pokrývala širokou škálu činností a akcí. Hlavní myšlenky jsou rozvedeny v těchto podkapitolách:

- Návrh a implementace řídícího systému pro automatizaci VLAN

- Správa závislostí
- Testování spolehlivosti nasazování sítě
- Správa uživatelských účtů a oprávnění
- Optimalizace v Ansible přístupu
- Možná budoucí rozšíření

6.2.1 Návrh a implementace řídícího systému pro automatizaci VLAN

Veškeré realizované části implementace řídícího systému pro automatizaci nasazování správy VLAN pro složité sítě byly provedeny na základě analýzy dané problematiky. Výstup diplomové práce umožňuje dynamické přidávání a odebírání jednotlivých (po jedné) VLAN v rozsáhlých sítích. Nástroj pracuje na základě podchytávání souvislostí o síti a provádí složitý systém operací, které je nutné provést pro vykonání této úlohy. Operace pro odebírání je ještě složitější, protože je nutné podchytit nutnost existence VLAN pro další systémy. Systém je založen na principech, které nevyžadují od lidské obsluhy nutnost vysoké interakce se systémem.

6.2.2 Správa závislostí

Důležitým bodem této práce bylo podchycení a navázání jednotlivých vazeb mezi síťovými prvky mezi sebou. Cílem je možnost využít této schopnosti pro provádění složitějších operací pomocí nástroje, který jim umožní jednoduše zadávat operace pro provádění těchto úkonů. Tento nástroj je důležitý pro usnadnění a zlepšení správy rozsáhlých sítí. Během práce byly vyvinuty mechanismy a nástroje pro identifikaci a podchycení vazeb pro řízení závislostí mezi jednotlivými vrstvami, síťovými prvky a organizační strukturou. Výhodné jsou i pro napojení do monitorovacího nástroje pro výkon sítě.

6.2.3 Správa uživatelských účtů a oprávnění

Další jmenovanou částí diplomové práce je nástroj pro distribuovanou správu uživatelských účtů na síťových zařízeních. Základním a jediným úkonem je vytváření a přepisování uživatelských účtů vedených v source-of-truth do jednotlivých zařízení. Správa uživatelských účtů je klíčovou součástí zajištění bezpečnosti a integrity sítě. Výhodou je možnost centrálně definovat a

ověřovat funkčnost přístupových práv pomocí testů a určovat z centrálního bodu, které účty budou založeny centrálně ve všech zařízeních.

6.2.4 Optimalizace v Ansible přístupu

Na základě získaných poznatků z vývoje práce a analýzy bylo možné provést řadu optimalizací. Ze začátku byly do Ansible propisovány jen změny, které se mají provést. Poté se v další verzi do Ansible přidaly moduly, které umožnily, aby výsledek diplomové práce načítal i část proměnných ze souboru, který získal z MongoDB. Poslední a finální verze byla připravena na modularitu a standardizaci vstupních formátů z MongoDB a prochází veškeré operace v rámci LOOP smyček. Navíc byly přidány podmínky, které vykonávají jen příkazy, které mají nějaké parametry. Tedy služby zkonfigurované jsou uvedeny do stavu uvedeném v source-of-truth tak, jak byly připraveny pro ETL pumpu. Následně jsou provedeny veškeré operace spojené s nahráváním. I zde došlo k optimalizaci na úrovni ukládaných dat. Dříve bylo v databázi vše ukládáno způsobem, který nebyl v souladu se žádnými standardy. Další verze navíc umožnily vkládat data dle předdefinované struktury. Zároveň se počítá s tím, že veškeré operace jsou na cluster z MongoDB, který umožňuje Trigger operace pro práci nad strukturami. To umožnilo standardizace a efektivní správu jednotlivých druhů složitějších operací.

6.2.5 Možná budoucí rozšíření

Diplomová práce také nastiňuje možné cesty budoucích výzkumných směrů v této oblasti (automatizace a správy počítačové sítě). Při rozšiřování je možné pro testování a vývoj použít jako simulátor jinou platformu.

7 Závěr

Při tvorbě diplomové práce byla provedena analýza dostupných zdrojů ke studovanému tématu. Následně byla pro prostudována problematika automatizované správy počítačových sítí (NetDevOps) - zvláště práce a distribuce VLAN nad dostupnou infrastrukturou spolu s příslušnými návaznostmi. Po analýze možných způsobů provedení práce byly vybrány metody implementace. Cenným zdrojem informací při studiu a analýze problematiky byla kniha *Automate your network* od Johna Capobianca. Na základech této knihy je vystavena celá diplomová práce.

Na základě analýzy vznikl nástroj, který má za úkol podchytit a zohlednit návaznosti jednotlivých konfiguračních komponent sítě. Nástroj pro jednotnou správu a nasazování nových verzí sítě používá parametrizované konfigurační šablony. Výsledek této diplomové práce byl následně zakomponován (s vhodnou skladbou open-source nástrojů) do CI/CD platformy hostované na CIV. Celkové řešení je v souladu s nastavenými pravidly pro bezpečnost nasazení konfigurace.

Výsledné řešení bylo otestováno na virtualizované síti v GNS3 za pomocí nástrojů Ansible, Ansible Molecule a PyATS. Řešení splnilo veškeré vstupní požadavky a bylo předvedeno vedoucímu práce.

Diplomová práce splnila veškeré body zadání, výsledkem je robustní a rozšiřitelný nástroj pro potřeby automatizovaného nasazování správy síťové infrastruktury v prostředí Gitlab CI/CD. Výsledné řešení bylo v návaznosti na nastudovanou problematiku vhodným způsobem otestováno.

Seznam obrázků

Seznam ukázek kódu

Seznam použitých zkratek

CI/CD Continuous Integration / Continuous Deployment

IaC Infrastructure as a Code

VLAN Virtual Local Area Network

WAN Wide Area Network

DevOps Developer + Operation

NetOps Network + Operation

ETL Operace Extrakce - Transformace - Load (nahrání)

.CFG Přípona konfiguračních souborů Cisco

.VLAN Přípona pro konfiguračních soubory VLAN

.VLAN Typ databáze (Not Only Structured Query Language = nejen SQL)

ORM Object realtion Mapping

CIV Centrum informatizace a výpočetní techniky

RIP Routing Information Protocol

BFS Breadth-first search

ARM Advanced RISC Machines

ICT Informační a komunikační technologie

CSV Comma separated values

ISP Internet service provider

ISO/OSI International Organization for Standardization/Open Systems Interconnection

QoS Quality of Services

SCP Secure Copy Protocol

RFC Request for Comments - norma

SSH Secure Shell

IoT Internet-of-Things

YAML Yet Another Markup Language

CLI Command Line Interface

CRON Plánovač úloh

API Application programming interface

REST Representational state transfer API

RAM Random access memory

pyATS Python Automation Testing System

INI Initialization - souborový formát

CPU Procesor

AWS Amazon Web Services

GCP Google Cloud Computing

Init Inicializace

JSON JavaScript Object Notation

DoS Denial of Services

DDoS Distributed Denial of Services

UPS Uninterruptible Power Supply/Source - zdroj nepřerušeného napájení (záloha)

IPv4 Internet Protocol version 4

DNS Domain Name System

XOps X + Operations

YANG Yet Another Next Generation - model

RPC Remote Procedure Calling

SNMP Simple Network Management Protocol

MIB Management information base - protokol

PDP Policy decision point

PEP Policy enforcement point

OPA Open policy agent

REGO Programovací jazyk pro OPA

TDD Test driven development

SDK Software development kit

IOS Operační systém pro Cisco switche

NX-OS Operační systém pro Cisco routery

CCNA Cisco Certified Network Associate

CCNP Cisco Certified Network Professional

SŘBD Systém řízení báze dat - databáze

TCP Transmission Control Protocol

UDP User Datagram Protocol

UP Port ve stavu propouštím nebo uplink

DWN Port ve stavu nepropouštím nebo downlink

Literatura

[1] Implementation of IPsec-VPN Tunneling using GNS3. *Institute of Advanced Engineering and Science*. 2017, Vol 7, No 3: September 2017.

[2] Personal Area Network. Dostupné z: `https://www.wikiwand.com/en/Personal_area_network`. Online; přístup dne 14. května 2024.

[3] Al-Hemairy, M. – Amin, S. – Trabelsi, Z. Towards more sophisticated ARP Spoofing detection/prevention systems in LAN networks. In *2009 International Conference on the Current Trends in Information Technology (CTIT)*, s. 1–6, 2009. doi: 10.1109/CTIT.2009.5423112.

[4] AWS, A. *NetDevOps: A Modern Approach to AWS Networking Deployments* [online]. 2023. Dostupné z: `https://aws.amazon.com/blogs/networking-and-content-delivery/netdevops-a-modern-approach-to-aws-networking-deployments/`.

[5] Capobianco, J. *Automate Your Network*. SanDiegoZoo, 2023/07/01.

[6] Choi, B. – Medina, E. *Is Ansible Good for Network Automation?*, s. 3–30. Apress, Berkeley, CA, 2023. doi: 10.1007/978-1-4842-9624-0_1. Dostupné z: `https://doi.org/10.1007/978-1-4842-9624-0_1`. ISBN 978-1-4842-9624-0.

[7] Emiliano, R. – Antunes, M. Automatic network configuration in virtualized environment using GNS3. In *2015 10th International Conference on Computer Science Education (ICCSE)*, s. 25–30, 2015. doi: 10.1109/ICCSE.2015.7250212.

[8] Ersoy, C. – Panwar, S. Topological design of interconnected LAN/MAN networks. *IEEE Journal on Selected Areas in Communications*. 1993, 11, 8, s. 1172–1182. doi: 10.1109/49.245906.

[9] ForcePoint. *What is the OSI Model?* [online]. ForcePoint. [cit. 2023/08/02]. Dostupné z: `https://www.forcepoint.com/cyber-edu/osi-model`.

[10] *Gtihub Cisco DevNet - Hands-on with NetDevOps* [online]. Github CISCO, 2016. [cit. 2024/03/09]. Cisco Campus DevNet Documentation. Dostupné z: `https://github.com/juliogomez/netdevops`.

[11] Golightly, L. – Modesti, P. – Chang, V. Deploying Secure Distributed Systems: Comparative Analysis of GNS3 and SEED Internet Emulator. *Journal of Cybersecurity and Privacy*. 2023, 3, 3, s. 464–492. doi: 10.3390/jcp3030024.

[12] Huawei. *What is YANG?* [online]. 2023. Dostupné z: `https://info.support.huawei.com/info-finder/encyclopedia/en/YANG.html`.

[13] La Red Martínez, F. D. L. A. *ISO/OSI model and data communication by animations* [online]. 2014. Online; přístup dne 14. května 2024. Dostupné z: `https://repositorio.unne.edu.ar/handle/123456789/30808`.

[14] *Testing Ansible Automation with Molecule Pt. 2* [online]. Medium, 2024. [cit. 2024/5/2]. Dostupné z: `https://medium.com/contino-engineering/testing-ansible-automation-with-molecule-pt-2-7e2ff5a70bcc`.

[15] *Introducing Ansible Molecule with Ansible Automation Platform* [online]. RedHat Inc., 2022. [cit. 2024/5/2]. Dostupné z: `https://developers.redhat.com/articles/2023/09/13/introducing-ansible-molecule-ansible-automation-platform`.

[16] *Developing and Testing Ansible Roles with Molecule and Podman - Part 1* [online]. Ansible, 2020. [cit. 2024/5/2]. Dostupné z: `https://www.ansible.com/blog/developing-and-testing-ansible-roles-with-molecule-and-podman-part-1/`.

[17] *Getting Started With Molecule* [online]. Medium, 2020. [cit. 2024/5/2]. Dostupné z: `https://ansible.readthedocs.io/projects/molecule/getting-started/`.

[18] Mysari, S. – Bejgam, V. *Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible* [online]. 2020.

[19] Neumann, J. C. *The Book of GNS3: Build Virtual Network Labs Using Cisco, Juniper, and More*. No Starch Press, 2015. ISBN 9781593275549.

[20] *Open Policy Agent: A Powerful Policy Engine for Cloud-Native Environments* [online]. Medium, 2023. [cit. 2024/4/1]. Dostupné z: `https://medium.com/@onixoni72/open-policy-agent-a-powerful-policy-engine-for-cloud-native-environments-8b0a1581`

[21] *What is OPA? Open Policy Agent Examples Tutorial* [online]. SpaceCraft, 2024. [cit. 2024/5/1]. Dostupné z: `https://spacelift.io/blog/what-is-open-policy-agent-and-how-it-works`.

[22] *How Do I Test Policies?* [online]. OpenPolicyAgent, 2024. [cit. 2024/5/2]. Dostupné z: https://www.openpolicyagent.org/docs/v0.12.2/how-do-i-test-policies/#data-mocking.

[23] Poth, A. – Werner, M. – Lei, X. How to Deliver Faster with CI/CD Integrated Testing Services? In *Systems, Software and Services Process Improvement*, 896 / *Communications in Computer and Information Science*, s. 401–409. Springer, 2018. doi: 10.1007/978-3-319-97925-0_33. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-319-97925-0_33.

[24] *Test-Driven Automation with Cisco pyATS Using SSH* [online]. github.io, 2022. [cit. 2024/1/17]. Dostupné z: https://ciscolearning.github.io/cisco-learning-codelabs/posts/pyats-ssh/#4.

[25] Rangnau, T. et al. Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, s. 145–154, 2020. doi: 10.1109/EDOC49727.2020.00026.

[26] RFC. RFC7223. RFC 7223, RFC Editor, 2014. Dostupné z: https://www.rfc-editor.org/info/rfc7223.

[27] Singh, C. et al. Comparison of Different CI/CD Tools Integrated with Cloud Platform. In *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, s. 7–12, 2019. doi: 10.1109/CONFLUENCE.2019.8776985.

[28] Svabik, J. *Rozdělení Sítí Podle Rozsahu (Pan, Lan, WAN) – kritéria členění ...* [online]. [cit. 2023/08/21]. Dostupné z: https://jansvabik.cz/matur/postvy/13.pdf.

[29] Wilfred A Melendez, E. L. P. The upper layers of the ISO/OSI reference model (part I). *Elsevier*. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/092054898690067X.

[30] Yang, Z. et al. Software-Defined Wide Area Network (SD-WAN): Architecture, Advances and Opportunities. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, s. 1–9, 2019. doi: 10.1109/ICCCN.2019.8847124.

[31] Zampetti, F. et al. CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, s. 471–482, 2021. doi: 10.1109/ICSME52107.2021.00048.

[32] Zezulka, F. Průmyslový Ethernet II: Referenční model ISO/OSI. *Elektrorevue*. 2014, 1, 1, s. 1–6. Dostupné z: https://www.researchgate.net/profile/F-Zezulka-2/publication/266604929_Prumyslovy_Ethernet_II_Referencni_model_ISOOSI/links/5885f6c2aca272b7b44ca312/Prumyslovy-Ethernet-II-Referencni-model-ISO-OSI.pdf.

Přílohy

Ukázka kódu 7.1: Zmenšená ukázka struktur, ve kterých jsou nahrané konfigurace jednotlivých síťových prvků

```
[{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet0/0",
        "type": "iana-if-type:ethernetCsmacd",
        "link-up-down-trap-enable": "enabled",
        "admin-status": "up",
        "enabled": true,
        "if-index": 1,
        "statistics": {
          "discontinuity-time": "1970-01-01T00
            ↪ :00:00+00:00"
        },
        "oper-status": "unknown",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.168.10.12",
              "netmask": "255.255.255.0",
              "origin": "static"
            }
          ]
        },
        "description": "HLR"
      },
    ]
  },
  "hostname": "CIV",
  "vlan": [
    {
      "vlan": "2",
      "interface": "Fa0/0",
      "ip": "192.168.1.1",
      "status": "up",
      "protocol": "up"
```

```
    },
    {
      "vlan": "3",
      "interface": "Fa0/1",
      "ip": "10.0.0.1",
      "status": "down",
      "protocol": "down"
    },
    {
      "vlan": "4",
      "interface": "Serial0/0",
      "ip": "172.16.1.1",
      "status": "up",
      "protocol": "up"
    }
  ]
}]
```

Ukázka kódu 7.2: Ukázkový konfigurační soubor Cisco - velmi zjednodušený

```
! Last configuration change at 13:14:54 UTC Wed Feb 21
   ↪ 2024
upgrade fpd auto
version 15.3
service timestamps debug datetime msec
service timestamps log datetime msec
service password-encryption
!
hostname CIV
!
boot-start-marker
boot-end-marker
!
aqm-register-fnf
!
enable secret 5 $1$PN2v$Gqfb7R3df9QoqJETGqWCf/
!
no aaa new-model
no ip icmp rate-limit unreachable
!
no ip domain lookup
ip domain name jarda.spos
ip cef
```

```
no ipv6 cef
!
multilink bundle-name authenticated
!
username admin password 7 10660C0A091843595F4A
!
redundancy
!
ip tcp synwait-time 5
ip ssh version 2
!
nterface GigabitEthernet0/0
 ip address 192.168.10.12 255.255.255.0
 description HLR
 duplex full
 speed 1000
 media-type gbic
 negotiation auto
 bridge-group 1
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
no cdp log mismatch duplex
!
control-plane
!
bridge 1 protocol ieee
bridge 1 route ip
!
mgcp behavior rsip-range tgcp-only
mgcp behavior comedia-role none
mgcp behavior comedia-check-media-src disable
mgcp behavior comedia-sdp-force disable
!
mgcp profile default
!
gatekeeper
 shutdown
!
line con 0
```

```
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line aux 0
 exec-timeout 0 0
 privilege level 15
 logging synchronous
 stopbits 1
line vty 0 4
 access-class 80 in
 login local
 transport input ssh
line vty 5 15
 login local
 transport input ssh
!
end
```

Ukázka kódu 7.3: Vstupní soubor pro popis operací nad sítí

```
{
 "data": {
   "datastore": "graph.db",
   "datasource": "file.db"
 },
 "global": {
   "scope": "staging"
 },
 "provider": {
   "endpoint": "${var.endpoint}",
   "username": "${var.username}",
   "password": "${var.password}"
 },
 "operation": {
   "type": "add",
   "subject": "vlan"
  },
 "parameters": {
   "vlan_id": "30",
   "vlan_name": "Test VLAN",

   "src_device": "CIV",
```

```
    "src_port": "GigabitEthernet1/0",

    "dst_device": "UN-A",
    "dst_port": "GigabitEthernet1/0"
  }
}
```

Ukázka kódu 7.4: YANG model pro interfaces v Cisco [26]

```
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |     +--rw name                          string
  |     +--rw description?                  string
  |     +--rw type                          identityref
  |     +--rw enabled?                      boolean
  |     +--rw link-up-down-trap-enable?     enumeration {
     ↪ if-mib}?
  |     +--ro admin-status                  enumeration {
     ↪ if-mib}?
  |     +--ro oper-status                   enumeration
  |     +--ro last-change?                  yang:date-and-
     ↪ time
  |     +--ro if-index                      int32 {if-mib
     ↪ }?
  |     +--ro phys-address?                 yang:phys-
     ↪ address
  |     +--ro higher-layer-if*              interface-ref
  |     +--ro lower-layer-if*               interface-ref
  |     +--ro speed?                        yang:gauge64
  |     +--ro statistics
  |        +--ro discontinuity-time    yang:date-and-
     ↪ time
  |        +--ro in-octets?            yang:counter64
  |        +--ro in-unicast-pkts?      yang:counter64
  |        +--ro in-broadcast-pkts?    yang:counter64
  |        +--ro in-multicast-pkts?    yang:counter64
  |        +--ro in-discards?          yang:counter32
  |        +--ro in-errors?            yang:counter32
  |        +--ro in-unknown-protos?    yang:counter32
  |        +--ro out-octets?           yang:counter64
  |        +--ro out-unicast-pkts?     yang:counter64
  |        +--ro out-broadcast-pkts?   yang:counter64
```

```
|        +--ro out-multicast-pkts?   yang:counter64
|        +--ro out-discards?         yang:counter32
|        +--ro out-errors?           yang:counter32
x--ro interfaces-state
   x--ro interface* [name]
      x--ro name                    string
      x--ro type                    identityref
      x--ro admin-status            enumeration {if-mib}?
      x--ro oper-status             enumeration
      x--ro last-change?            yang:date-and-time
      x--ro if-index                int32 {if-mib}?
      x--ro phys-address?           yang:phys-address
      x--ro higher-layer-if*        interface-state-ref
      x--ro lower-layer-if*         interface-state-ref
      x--ro speed?                  yang:gauge64
      x--ro statistics
         x--ro discontinuity-time    yang:date-and-
            ↪ time
         x--ro in-octets?            yang:counter64
         x--ro in-unicast-pkts?      yang:counter64
         x--ro in-broadcast-pkts?    yang:counter64
         x--ro in-multicast-pkts?    yang:counter64
         x--ro in-discards?          yang:counter32
         x--ro in-errors?            yang:counter32
         x--ro in-unknown-protos?    yang:counter32
         x--ro out-octets?           yang:counter64
         x--ro out-unicast-pkts?     yang:counter64
         x--ro out-broadcast-pkts?   yang:counter64
         x--ro out-multicast-pkts?   yang:counter64
         x--ro out-discards?         yang:counter32
         x--ro out-errors?           yang:counter32
```

Ukázka kódu 7.5: Trigger v MongoDB Atlas

```
exports = async function(changeEvent) {

  const docId = changeEvent.documentKey && changeEvent.
     ↪ documentKey._id;

  const serviceName = "Cluster0";
  const sourceDatabase = "diplomka";
  const destDatabase = "diplomka";
  const collectionSrc = "diplomka";
```

```
  const collectionDst = "backup";
  const sourceCollection = context.services.get(
    ↪ serviceName).db(sourceDatabase).collection(
    ↪ collectionSrc);
  const destCollection = context.services.get(
    ↪ serviceName).db(destDatabase).collection(
    ↪ collectionDst);

  try {
    if (changeEvent.operationType === "update" ||
      ↪ changeEvent.operationType === "replace" ||
      ↪ changeEvent.operationType === "delete") {
      const oldItem = await sourceCollection.findOne({"
        ↪ _id": docId});
      await destCollection.insertOne(oldItem);
    }
  } catch(err) {
    console.log("Chyba pri zpracovani zmeny: ", err.
      ↪ message);
  }
};
```

Ukázka kódu 7.6: Gitlab pipeline definice

```
variables:
  ENVIRONMENT:
    description: "Environment to deploy to"
    value: "production"

default:
  image: cytopia/ansible:2.12


stages:
  - start
  - operation
  - ansible
  - deploy

etl:
  stage: start
  image: alpine:3.17
  script:
```

```
    - "python3 main.py init"

etl-delete:
  stage: start
  image: alpine:3.17
  script:
    - "python3 main.py delete"
  rules:
    - when: manual


test:
  stage: test
  image: alpine:3.17
  script:
    - "python3 main.py validate"


generate_anisble_files:
  stage: ansible
  image: alpine:3.17
  script:
    - "python3 main.py all"


deploy-network:
  stage: deploy
  before_script:
    - which ssh-agent || ( apk --update add openssh-
      ↪ client )

    - eval $(ssh-agent -s)


  ## Create the SSH directory and give it the right
     ↪ permissions

    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - chmod 644 ~/.ssh/known_hosts

  script:
```

```
60      - "ansible-playbook -i ansible/inventory ansible/
            ↪ playbooks/my_playbook.yml"
61    environment: $ENVIRONMENT
62    ## Manually or when it is started by another pipeline
63
64    rules:
65      - if: $CI_PIPELINE_SOURCE == "pipeline"
66      - when: manual
```

Uživatelská příručka

V této příručce je popsáno jak provozovat bakalářskou práci. Je uveden popis technologií potřebných pro běh práce.

Technologie

Pro hladký běh programu je předpokládána instalace následujících technologií:

- `Ansible` - nástroj pro provádění operací nad switchi
- `Anaconda3` - distribuce balíčků Python, R pro vědecké výpočty
- `Jupyter Notebook` - webové běhové prostředí pro Python
- `pip` - Balíčkovací manager
- `ttp_templates` - Balíčkovací manager
- `Python` 3 - programovací jazyk

Pro běh jsou potřeba také tyto balíčky:

- `numpy` - knihovna v Pythonu poskytující infrastrukturu pro práci s vektory, maticemi a obecně vícerozměrnými poli
- `graphx` - knihovna v Pythonu umožňující práci s grafem pro uložení vazeb mezi jednotlivými vrstvami sítě
- `scipy` - poskytuje v Pythonu algoritmy například pro optimalizace, integrace, interpolace, albegraické výpočty, diferenciální výpočty, statistiku i pro grafy
- `matplotlib` - knihovna umožňující statické, anomované a interaktivní vizualizace
- `tqdm` - progress-bar pro vizualizaci průběhu činnosti
- `plot_utils` - generátor 2D grafik na základě dat
- `pytest` - knihovna pro testování v Pythonu

Možnou alternativou pro běh je `Google Colab`, který běží kompletně v cloudovém prostředí (na externí výpočetní infrastruktuře připravené pro běh specializovaných aplikací).

Sestavení programu

Program by mělo být možné spouštět na všech operačních systémech (pip, python3 i anaconda jsou dostupné pro Linux, MacOS i Windows). Vývoj však probíhal a byl testován pouze na Windows. Program běží v příkazové řádce a využívá se služeb `pip` pro instalaci potřebných závislostí. Instalace závislostí probíhá manuální přes requirements.txt. Program využívá pro načtení dat, vizualizaci a ověření výsledků.

Příprava složek

Pro potřeby fungování je nutné dodržet následující schéma složek, jak je uvedeno v adresářové struktuře dále.

Distribuce VLAN - add/delete

Pro správné fungování distribuce vlan, je potřebné mít ve vstupním souboru input.json parametry, jak je uvedeno v následující ukázce:

Ukázka kódu 7.7: Vstupní soubor pro popis operací nad sítí

```
{
 "data": {
   "datastore": "graph.db",
   "datasource": "file.db"
 },
 "global": {
   "scope": "staging"
 },
 "provider": {
   "endpoint": "${var.endpoint}",
   "username": "${var.username}",
   "password": "${var.password}"
 },
 "operation": {
   "type": "add",
   "subject": "vlan"
  },
 "parameters": {
   "vlan_id": "30",
   "vlan_name": "Test VLAN",

   "src_device": "CIV",
```

```
    "src_port": "GigabitEthernet1/0",

    "dst_device": "UN-A",
    "dst_port": "GigabitEthernet1/0"
  }
}
```

V případě potřeby odebírání VLAN po trase se změní operation.type na delete.

Spouštění celkového systému

Ve složce ve které je soubor main.py se spustí tento příkaz:

`python main.py <fáze ke spuštění>`

Fáze systému jsou: all, validate, init a full. Pro spuštění celkového skriptu je potřeba zvolit fázi ALL. Pro validaci jen část validate. Pro inicializaci init a jen pro generování souborů fázi full. Příkaz spouští jednotlivé fáze, které chceme.

Výsledkem poslední fáze je generování souborů pro Ansible. Ansible playbook je následně spuštěn pomocí tohoto příkazu:

`ansible-playbook -i inventory <nazev-playbooku.yml>`

Spouštění testů

Najdeme si příslušné soubory ve složce test a příslušných složkách.

`python3 test_utils.py`

`python3 cmd_helper_test.py` apod.

Deploy uživatelů

Deploy uživatelů je připraven pro nasazení uživatelských účtů přímo do všech switchů definovaných v inventory. Pro spuštění celé operace je nejdříve nutné nastavit soubor s definicí jednotlivých uživatelů.

Ukázka kódu 7.8: Soubor s definicí jednotlivých uživatel

```
users:
  - username: user1
    password: password1
  - username: user2
    password: password2
```

Předpokládá se, že veškeré účty budou mít ty nejvyšší oprávnění (privilege 15). Je tedy možné nad těmito účty pouštět Ansible skripty (hlavní důvod realizace).

Poté je nutné spustit příkaz pro deploy uživatel na switche.

`ansible-playbook -i inventory deploy_user.yml`

Uživatelé jsou následně vytvořeny. Pro připojení k těmto uživatelům je možné využít následující příkaz. Po jeho spuštění budete vyzváni k zadání definovaného hesla.

Pro přístup se využívá protokolu SSH, který musí být na switchi nakonfigurován.

`ssh user1@IP_ADRESA_SWITCHE -oKexAlgorithms=+diffie-hellman-group1-sha1 -oCiphers=aes256-cbc`

Příprava switche

Nejdříve je nutné na switchi vytvořit IP Adresu, aby bylo se ke switchi připojit.

Ukázka kódu 7.9: Nastavení IP adres

```
conf t
int GigabitEthernet0/0
ip address 192.168.10.11 255.255.255.0
no sh
end
```

Následně se vraťte z enable módu. Pro ověření nastavení IP adresy je možné použít příkaz: `sh ip int br`.

Poté je nutné vytvořit uživatele pro přístup

Ukázka kódu 7.10: Nastavení defaultního uživatele

```
conf t
username admin password MojeHeslo
enable secret MojeHeslo
line vty 0 4
login local
end
w
write memory
```

Poté je možné použít jednoduchý Ansible skript pro ověření funkčnosti show_clock.yml.

8 Obsah elektronické přílohy

8.1 Adresářová struktura

- `/`
  - `Poster` – složky s posterem
  - `Aplikace_a_knihovny`
    - `db_loader.py` – python skript pro nahrávání informací do relační databáze
    - `example.log` – logovací soubor pro aplikaci
    - `test` – soubory testů
    - `graph.db` – relační databáze pro uchovávání grafových struktur
    - `graph.py` – skript pro práci s grafem - jednoduché operace nad obecným grafem
    - `graph_builder.py` – skript pro tvorbu grafu ze vstupních informací - podchycení jednotlivých vazeb a tvorba potřebných návazností -> tvorbu struktur logické a fyzické sítě
    - `input.json` – konfigurační soubor pro stanovení parametrů
    - `load_logical_structure.py` – nástroj pro tvorbu transformaci a návazností na ukládání logické struktury
    - `main.py` – hlavní bod programu
    - `to_dict.py` – převod jednotlivých stuktur na dictionary
    - `params.json` – parametry pro input.json
    - `vars.json` – externí proměnné pro input.json
    - `ansible` – veškeré soubory pro běh Ansible skriptů
    - `data` – složky obsahuje jednotlivé konfigurac Cisco prvků a soubory s VLAN výpisem
    - `data-inserted` – ID jednotlivých zařízení vložených do logicko-fyzické vrstvy
    - `datapump` – datová pumpa pro nahrávání dat do MongoDB
    - `generator` – soubory pro potřeby generování příslušných inventory, hosts pro Ansible
    - `logical` – množina souborů pro práci s logickou vrstvou modelu
    - `network` – práce s grafem sítě
    - `operation` – implementace jednotlivých operací pro práci nad sítí
    - `utils` – jednoduché utils nástroje pro obecnou práci s daty
    - `yang` – nástroje pro práci s YANG modelem
  - `Text_prace`
    - `tex` – zdrojové soubory TEXvčetně přiložených souborů
    - `A22N0054P` – PDF soubor bakalářská práce
  - `readme.txt` – popis adresářové struktury