VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020

Bc. Maroš Matisko

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NÁSTROJ PRO GENEROVÁNÍ NÁHODNÉ KONFIGURACE KYBERNETICKÉ ARÉNY

A TOOL FOR GENERATING A RANDOM CONFIGURATION OF A CYBER ARENA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Maroš Matisko**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Václav Uher, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Maroš Matisko ***ID:*** 185932
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Nástroj pro generování náhodné konfigurace kybernetické arény

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit generátor konfigurace pro nástroj automatizace konfigurace (například Ansible) podle předdefinovaných kritérií (různé rozsahy podsítí a pevné IP adresy, jiné rozsahy portů, atd.) Aplikace bude vyvíjena v jazyce Python a bude navržena tak, aby bylo možné přidat podporu pro další konfigurační nástroje. Dále bude aplikace podporovat vygenerování více verzí scénáře jedním spuštěním. Výstup bude následně ověřen nasazením vygenerované konfigurace pro vybraný scénář a otestováním jeho funkčnosti.

DOPORUČENÁ LITERATURA:

[1] EDELMAN, Jason, Scott LOWE a Matt OSWALT. Network programmability and automation: skills for the next-generation network engineer. Sebastopol, California: O'Reilly Media, 2018. ISBN 978-149-1931-257.

[2] BEYER, Betsy, Chris JONES, Jennifer PETOFF a Niall Richard MURPHY. Site reliability engineering: how Google runs production systems. Boston: Oreilly, 2016. ISBN 14-919-2912-X.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Václav Uher, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práca sa zameriava na návrh a implementáciu nástroja pre generovanie konfigurácie určenú pre automatizačný nástroj Ansible. Výstupom nástroja je vygenerovaná konfigurácia, ktorá obsahuje náhodné hodnoty podľa určených pravidiel a bola nasadená na virtuálnej testovacej infraštruktúre. Teoretická časť práce popisuje prístupy sieťovej automatizácie pri nasadzovaní a konfigurácii cieľových zariadení a ich nasadenie v procese nazývanom Infraštruktúra ako kód. Bližšie popisuje nástroj Ansible, ktorý bude využívať výstup praktickej časti práce. Praktická časť práce sa zameriava na návrh a implementáciu generátora konfigurácie pre nástroj Ansible a prezentuje výsledky testovania generátora a vygenerovanej konfigurácie.

KĽÚČOVÉ SLOVÁ

Ansible, generátor, Infraštruktúra ako kód, konfigurácia, Python, sieťová automatizácia

ABSTRACT

The master's thesis is focused on the design and implementation of a tool for generating configuration named Ansible. The result of using this tool is generated configuration, which contains random values chosen according to specified parameters and it was deployed on a virtual testing infrastructure. The theoretical part describes approaches of network automation in the process of deploying and configuration of network devices called Infrastructure as code. It also describes programme Ansible, which will be using the output of the implemented tool. The practical part of the thesis is focused on designing the sfunctionality and internal structure of the tool, implementation of the tool and testing implemented tool as well as generated configuration.

KEYWORDS

Ansible, configuration, generator, Infrastructure as code, network automation, Python

MATISKO, Maroš. *Nástroj na generování náhodné konfigurace kybernetické arény*. Brno, Rok, 67 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Václav Uher, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „Nástroj na generování náhodné konfigurace kybernetické arény“ som vypracoval samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno . . . .

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Václavovi Uhrovi, PhD. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Jedným z trendov posledných rokov je postupná centralizácia výpočtového výkonu do tzv. dátových centier (skrátene datacentier). Z hľadiska úspory nákladov či priestoru to prináša viaceré výhody, no okrem toho, že to otvára nové možnosti využitia zariadení, prináša to aj nové výzvy z hľadiska bezpečnosti, či prevádzky vzdialených zariadení v týchto dátových centrách.

Správa veľkej skupiny zariadení v rámci jedného objektu, organizácie a pod. však prináša administrátorom podobné výzvy. Medzi výzvy možno zaradiť fyzickú náročnosť presunu medzi jednotlivými zariadeniami, časovú náročnosť samotnej správy zariadení a iné. Najmä časová náročnosť správy veľkého množstva zariadení, ktoré navyše nemusia byť fyzicky dostupné, viedla postupom času k zavedeniu sieťovej automatizácie.

Sieťovou automatizáciou možno označiť proces správy, dohľadu a kontroly zariadení, ktorý je geograficky nezávislý od fyzického umiestnenia cieľového zariadenia. Tento proces má vopred definované akcie, ktoré sa zvyčajne opakujú, čím šetria čas administrátora. V tomto procese sú využívané rôzne sieťové protokoly alebo nástroje, ktoré tieto protokoly zvyčajne využívajú. Toto riešenie má aj ďalšie výhody, príkladom čoho môže byť napr. aj štatistika využitia siete, ktorú je možné využiť pre efektívnejšie manažérske rozhodnotia a pod.

Témou diplomovej práce je vývoj generátora konfigurácie pre existujúci nástroj sieťovej automatizácie. Daný nástroj má byť schopný prispôsobiť šablónovú konfiguráciu zadaným kritériám a táto konfigurácia má byť následne overená praktickým spustením daného nástroja.

Hlavným prínosom tejto práce je vytvorenie generátora konfigurácie pre nástroj sieťovej automatizácie na základe analýzy konfigurácie používanej zvoleným nástrojom sieťovej automatizácie a demonštrácia jeho výstupu. Dve úvodné kapitoly sú venované teoretickej časti, na ktorej táto práca stavia. V nasledujúcich kapitolách je popísaný samotný vývoj a testovanie generátora

Prvá kapitola popisuje rozbor sieťovej automatizácie, prinášané výhody a najčastejšie používané nástroje a sieťové protokoly. Druhá kapitola sa venuje prístupu sieťovej automatizácie, nazývanej Infraštruktúra ako kód, ktorý je postavený na textovom zápise konfigurácie infraštruktúry, vrátane vytvorenia infraštruktúry, rôznych inštalácií systémov a programov, ich nastavení a pod. Tretia kapitola definuje požiadavky na generátor konfigurácie a rieši teoretický návrh generátora. Vo štvrtej kapitole je prezentovaná implementácia generátora v jazyku Python a úpravy od pôvodného návrhu. Piata kapitola obsahuje postup na testovanie a návrh virtuálnej infraštruktúry. V poslednej 6. kapitole sú prezentované výstupy jednotlivých režimov generátora a testovanie časovej náročnosti generovania konfigurácií.

1 Sieťová automatizácia

Administrácia siete sa skladá z procesov konfigurácie, správy, testovania a nasadzovania jednotlivých fyzických alebo virtuálnych prvkov siete. Tieto prvky siete môžeme rozdeliť na sieťové, ktoré prepájajú koncové zariadenia a starajú sa o prenos dát, a na koncové prvky, ktoré pre účely tohto rozdelenia možno definovať tak, že primárne generujú, posielajú a prijímajú dáta [1]. V prípade sieťových prvkov sa jedná najmä o smerovače a prepínače (routre a switche), v prípade koncových zariadení sú to najčastejšie pracovné stanice (workstations) alebo servery. So zväčšujúcimi sa sieťami a postupným rozširovaním softvérových technológií ako sú virtuálne clustre, softvérovo definové siete (SDN) a iné, je však administrácia čoraz časovo náročnejší proces. Tento problém rieši práve sieťová automatizácia jednotlivých procesov.

1.1 Výhody riešenia

Sieťová automatizácia jednotlivých procesov má viacero nesporných výhod, medzi ktoré mimo iného patria aj:

- efektivita – časté opakovanie jednotlivých úkonov v daných procesoch je možné automatizovať a spúšťať série príkazov [2],
- zníženie chybovosti – eliminácia preklepov vytvorením, otestovaním a následným spúšťaním série príkazov,
- zvýšenie prehľadnosti procesov – definované série príkazov zvyšujú prehľadnosť v jednotlivých procesoch,
- zníženie nákladov – kumuláciou vyššie uvedených výhod dochádza k úspore nákladov na vykonanie procesu.

1.2 Možnosti nasadenia

Sieťovú automatizáciu možno využiť pri takmer všetkých procesoch administrácie siete [3]. Výnimku tvoria akcie, ktoré sú svojou povahou unikátne, napr. špecifické nastavenia pre jedno zariadenie, či riešenie vzniknutého problému. Medzi scenáre, v ktorých je vhodné využiť výhody sieťovej automatizácie možno zaradiť napr.:

- Nasadenie koncových zariadení vyhradených na spoločný účel – príkladom môže byť skupina webových serverov, medzi ktoré bude rozložená záťaž, takže zdieľajú rovnaké nastavenia, odlišné nastavenia sú riešené konkrétnym zápisom pre každý server, ktoré môže byť uložené napr. v liste názvov pre servery.
- Konfigurácia zariadení – zmena nastavení platná pre skupinu zariadení, napr. zmenou organizácie či topológie siete, alebo celkovou inováciou siete.

- Dohľad nad sieťou – jednotlivé zariadenia o svojom stave informujú a hlásia mimoriadne stavy prostredníctvom štandardizovaných protokolov, alebo pomocou nich umožňujú zachytávanie a analýzu prenášaných dát.
- Testovanie siete alebo jednotlivých zariadení v sieti – automatizácia umožňuje definovať scenáre, ktoré by mali nastať pri zadaní série príkazov a overenie, či tieto scenáre skutočne nastali. V prípade opaku umožňuje zaznamenať situáciu v sieti, ktorá môže byť neskôr analyzovaná a z nej následne môžu byť zistené nedostatky.

1.3 Výber vhodného automatizačného nástroja

V súčasnosti existuje väčšie množstvo automatizačných nástrojov, z ktorých nie je možné určiť všeobecne najlepšie riešenie bez informácii o konkrétnej sieti, v ktorej má byť riešenie využívané. Druhým faktorom výberu sú požadované funkcie automatizačného nástroja a taktiež dôležitým faktorom sú aj požiadavky samotného nástroja na jeho nasadenie. Najdôležitejším úkonom v procese výberu automatizačného nástroja je definícia požiadaviek na nástroj a analýza siete. V nasledujúcom kroku je to výber z dostupných existujúcich riešení [4]. Samotnú automatizáciu je často vhodné podporiť automatizáciou monitoringu resp. analýzy siete. Pre daný účel existuje niekoľko sieťových protokolov umožňujúcich práve tieto činnosti.

1.3.1 Existujúce nástroje

Sieťová automatizácia v priebehu posledných niekoľko rokov pokročila a postupne sa využíva aj v čoraz menších prostrediach, ktoré si uvedomujú jej výhody. Jedná sa o voľne šíriteľné, či komerčné nástroje, alebo v niektorých prípadoch aj samostatné vlastné skripty, ktoré zvyčajne plnia jeden alebo malé množstvo účelov. Pre takéto skripty sú vhodné moderné skriptovacie jazyky, akými sú napr. Python alebo Ruby, príp. skriptovacie jazyky podporované priamo daným operačným systémom, akými sú Bash a Powershell. Medzi najpoužívanejšie nástroje sieťovej automatizácie podľa Perkina[5] patria nástroje uvedené nižšie.

Ansible

Automatizačný nástroj od firmy Red Hat je vyvinutý v programovacom jazyku Python a bol vydaný v roku 2012 [5, 6]. Hlavnou výhodou Ansible je jeho univerzálnosť, každý používateľ môže napísať vlastný modul, pomocou ktorého rozšíri jeho využitie. Momentálne existuje viac ako 200 oficiálnych modulov, do tohto počtu patria aj moduly od známych spoločností, napr. Junos, F5, Cloudengine a ďalšie. Ďalšou

nespornou výhodou tohto nástroja je nasadenie bez nutnosti používať agentov na klientských zariadeniach, čo výrazne rozširuje zoznam zariadení, ktoré je takto možno konfigurovať. Z týchto dôvodov bol tento nástroj vybraný pre účely praktickej časti tejto práce a detailnejšie sa mu bude venovať kapitola 2.3.

Puppet

Open source automatizačný nástroj vyvinutý v jazyku Ruby v roku 2005 [5, 7]. Slúži primárne na konfiguráciu koncových zariadení, hlavným rozdielom oproti nástroju Ansible je nutnosť inštalácie agenta na klientskom zariadení. Podporuje ako koncové zariadenia (najmä servery postavené na OS Linux), tak aj sieťové zariadenia (napr. časť zariadení od firmy Cisco). Na konfiguráciu zariadení využíva vlastný konfiguračný jazyk, inšpirovaný jazykom programu Nagios. Samotná konfigurácia funguje na pull princípe, v ktorom si agent na zariadení, ktoré má byť nakonfigurované, vypýta od master servera konfiguráciu. Master server má uložené konfigurácie pre všetky zariadenia. Klient aj server sa voči sebe overujú certifikátom typu X.509. Výhodou je možnosť periodickej aktualizácie klientského zariadenia, prednastavená hodnota je 30 minút, po uplynutí ktorého sa overuje konfigurácia na klientskom zariadení.

Chef

Automatizačný nástroj založený na koncepte podobnom nástroju Puppet, taktiež napísaný v programovacom jazyku Ruby a vyžaduje inštaláciu agenta na klientskom zariadení [5, 8]. Na rozdiel od nástroja Puppet však navyše vyžaduje pracovnú stanicu na manažment master servera. Oba tieto nástroje podporujú tvorbu a nasadenie vlastných modulov v doménovo špecifickom jazyku založenom na jazyku Ruby, ktoré môžu zjednodušiť konfiguráciu klientských zariadení. Podporuje tzv. "suchý beh" (angl. `dry run`), ktorý umožňuje vopred otestovať konfiguráciu využitím virtualizácie, čo znižuje riziko chyby pri nasadení do produkčného systému.

SaltStack

Automatizačný nástroj určený predovšetkým na automatizáciu serverov, známy tiež pod názvom Salt [5, 9]. Funguje na topológii master – minions, v ktorej je tzv. `minion` ekvivalentom klasickej role klienta. Podobne ako dva vyššie uvedené nástroje vyžaduje inštaláciu agenta na klientskom zariadení, takže v prípade, ak dané zariadenie tohto agenta nepodporuje (napr. ak sa jedná o neštandardné zariadenie), je možné napísať vlastné rozhranie pre agenta. Samotný nástroj bol vyvinutý v jazyku Python a v tomto jazyku taktiež podporuje vlastné rozhrania.

Jenkins

Automatizačný nástroj používaný pre účely Continous Integration / Continous Delivery (CI/CD) [5, 10]. V praxi to môže znamenať situáciu, pri ktorej Jenkins dohliada na zmeny vo vývojovom nástroji, napr. Git serveri a v prípade zmeny kódu prevedie definované udalosti (zostavenie projektu pre rôzne platformy, spustenie testov a ich vyhodnotenie, zostavenie binárnych súborov, či nasadenie na cieľové zariadenie). Podporuje prídavné moduly, ktoré rozširujú základnú funkcionalitu nástroja. Dôležitou vlastnosťou je distribuovanosť, v praxi to znamená topológiu master – workers. S okolitým svetom komunikuje len master server, ktorý prijíma úlohy, rozdeľuje ich medzi worker servery, následne prijíma výsledky a informuje o výsledkoch zadávateľa úlohy.

1.3.2 Protokoly sieťovej automatizácie

Časť sieťovej automatizácie, primárne je to konfigurácia a dohľad nad sieťovými zariadeniami resp. analýza stavu siete, je založená na štandardizovaných aplikačných protokoloch, ktoré podporuje veľká časť výrobcov hardvéru a je možné použiť akýkoľvek softvér, ktorý podporuje daný protokol.

Simple Network Management Protocol (SNMP)

Aplikačný protokol určený na výmenu konfiguračných informácii jednotlivými sieťovými zariadeniami. Manažované zariadenia musia obsahovať funkciu SNMP agenta, ktorý im umožňuje týmto protokolom komunikovať s riadiacim prvkom – nazvaným Network Management System (NMS) [11]. Typicky sa jedná o pracovnú stanicu alebo server s riadiacim programom, ktorým je možné zasielať príkazy jednotlivým agentom. Agent prijíma konfiguračné príkazy od NMS, zhromažďuje informácie o svojom zariadení a zasiela signály NMS v prípade výskytu definovaných situácií, ktoré sú definované tzv. Trap pravidlami.

NetFlow

Protokol od firmy Cisco určený na zber informácii o prenášaných dátach a dohľad nad sieťou [12]. Jednotlivé sieťové prvky, obvykle smerovače, ktoré majú povolený NetFlow protokol zbierajú informácie o odoslaných dátach, z ktorých následne vytvárajú NetFlow záznamy. Tieto záznamy následne spracováva a vyhodnocuje server, do ktorého sú zasielané. Výsledkom je sieťová analýza, ktorú analyzér prezentuje správcovi v grafickej podobe. Na základe tohto protokolu vznikli ďalšie proprietárne protokoly iných firiem, ako napr. JFlow od firmy Juniper, s-flow od firiem 3Com/HP, Dell a Netgear, či Cflow od firmy Alcatel-Lucent.

OpenFlow

Protokol technológie Softvérovo definovaných sietí (SDN), pomocou ktorého je zabezpečená komunikácia medzi SDN kontrolérom a SDN sieťovými zariadeniami [13]. Prenáša pravidlá pre dané sieťové zariadenie, v smere od kontroléra ku ovládaným SDN prepínačom. Nad kontrolérom beží riadiaca SDN aplikácia, ktorá umožňuje vytváranie a zmenu smerovacích pravidiel, pomocou ktorých je riadená SDN sieť. Tým je dosiahnuté centralizované riadenie siete, čo znižuje výpočtové nároky na jednotlivé sieťové zariadenia. Prípadná zmena v konfigurácii je nutná len na jednom mieste v sieti, čo znižuje časovú náročnosť v prípade potreby zmien.

Secure Shell (SSH)

Aplikačný protokol určený na zabezpečené vzdialené pripojenie naprieč nezabezpečenou sieťou (náhrada protokolov typu telnet, rlogin a pod.)[14]. Týmto protokolom možno zasielať textové príkazy, či zabezpečene tunelovať iné aplikačné protokoly, napr. SCP alebo SFTP na prenos súborov. Okrem tejto viditeľnej funkcionality protokol ponúka obojstrannú autentizáciu pri vytváraní spojenia (overenie oboch strán – heslom, asymetrickým kľúčom či certifikátom), integritu prenášaných dát a transparentné šifrovanie (z pohľadu koncovej aplikácie). V kontexte sieťovej automatizácie sa využíva na pripojenie k vzdialeným serverom resp. pracovným staniciam, ktoré sú konfigurované alebo monitorované.

2 Infraštruktúra ako kód

Z anglického výrazu `Infrastructure as code` [15], v skratke IaC, je proces správy a nasadenia konfigurácie automatizovanou formou. Využíva nástroje sieťovej automatizácie, ktoré podľa použitia možno rozdeliť na dve väčšie skupiny a to nástroje na orchestráciu konfigurácie a nástroje na správu konfigurácie [16]. Toto rozdelenie nie je úplne presné, nástroje na orchestráciu konfigurácie často obsahujú niektoré funkcie správcov konfigurácie a to isté platí aj naopak.

Obe skupiny riešia veľmi podobné problémy vychádzajúce z rovnakej príčiny a tou je rôznorodosť vývojového prostredia [17]. Z toho vychádza základná vlastnosť IaC nástrojov, tzv. idempotencia (angl. `Idempotence`), ktorá popisuje schopnosť týchto nástrojov dosiahnuť rovnaký výsledný stav na danom zariadení bez ohľadu na jeho počiatočnú konfiguráciu.

Cieľom tejto práce je vytvoriť generátor konfigurácie pre automatizačný nástroj, patriaci do skupiny IaC nástrojov, Ansible. Nástroj je bližšie popísaný v podkapitole 2.3. Generátor konfigurácie bude generovať niekoľko verzií jedného scenára definovaného v šablóne. Tieto konfigurácie môžu byť využité pre počítačové cvičenia, kde sa každému študentovi vygeneruje scenár s odlišným rozsahom IP adries, odlišnými transportnými portami, heslami a pod. Táto kapitola preto poskytuje teoretický základ pre používanie tohoto nástroja a jeho jednotlivých častí. Samotný nástroj Ansible pracuje s konfiguráciou v textovom formáte YAML a pre vkladanie premenlivých, či citlivých dát využíva Jinja šablóny. Tieto formáty sú bližšie popísané v časti 2.3.1. V prípade prítomnosti citlivých dát je možné využiť šifrovanú šablónu či celú konfiguráciu pomocou nástroja Ansible Vault, viac uvádza časť 2.3.3.

2.1 Orchestrácia konfigurácie

Z anglického originálu `Configuration Orchestration` [18]. Jedná sa o nástroj, ktorý umožňuje automatické nasadenie infraštruktúry, zvyčajne na poskytnutých zdrojoch od poskytovateľa cloudových služieb (samostatné servery, CDN, sieťové prvky – load balancery, firewally, smerovače, prepínače a pod.). Príkladom takéhoto orchestračného nástroja je napr. Terraform, CloudFormation či Cloudify [19], príkladom poskytovateľov služieb je Amazon AWS, Microsoft Azure, Google Cloud.

Podstatnou črtou orchestračného nástroja v porovnaní so správcom konfigurácie je jeho nemennosť (angl. `immutability`), ktorá pri zmene konfigurácie vytvorí nový prvok alebo infraštruktúru namiesto zmeny tej aktuálnej. Výhody tohto prístupu sú najmä pri testovaní a nasadzovaní nových, či nie úplne stabilných technológií, ktoré sú nachylné na zmeny konfigurácie.

2.2 Správca konfigurácie

V anglickej literatúre uvádzaný ako `Configuration Manager` [18], je nástroj používaný na úpravu a aplikácie vytvorenej konfigurácie pre vzdialené zariadenie. Prehľad nástrojov používaných na správu konfigurácie možno nájsť v kapitole 1.3.1. V závislosti od používaného nástroja sú na konfiguráciu využívané štandardné alebo proprietárne protokoly pomocou ktorých prebieha pripojenie na vzdialené zariadenie a jeho konfigurácia. Príklady štandardných protokolov používaných pri správe konfigurácie sú uvedené v kapitole 1.3.2.

2.2.1 Prístupy správy konfigurácie

Na dosiahnutie požadovanej konfigurácie na vzdialenom zariadení využívajú nástroje na správu dva rozdielne prístupy, ktoré sa líšia v logike prístupu ku konfigurácii [20].

Procedurálny prístup

Procedurálny prístup definuje, ako sa majú jednotlivé kroky vykonávať a čo je ich parametrom. Môžeme povedať, že je to veľmi zjednodušená forma zápisu série príkazov krok za krokom. Výhodou je absolútna kontrola nad vykonávanými príkazmi, nevýhodou je časová náročnosť písania konfiguračných súborov. Tento spôsob využívajú napr. nástroje Ansible a Chef.

Deklaratívny prístup

Deklaratívny prístup spočíva v definícii stavu, ktorý chceme na vzdialenom zariadení dosiahnuť a nástroj na správu rieši, pomocou akých krokov a v akom poradí dosiahne daný stav. Výhodou je nižšia časová náročnosť, nevýhodou strata kontroly nad vykonávanými procesmi. Tento prístup používajú nástroje ako CloudFormation, Puppet, SaltStack či Terraform.

Porovnanie prístupov

Pre jednoduchšie pochopenie rozdielov medzi dvomi príkladmi slúži nasledujúci jednoduchý praktický príklad [21]: Na platforme AWS od Amazonu chceme nasadiť 10 virtuálnych serverov, ktoré budú obsluhovať aplikáciu. Procedurálny zápis vo formáte YAML pre nástroj Ansible by vyzeral nasledovne:

```
- ec2:
        count: 10
        image: ami-v1
        instance_type: t2.micro
```

Naproti tomu by príklad deklaratívneho prístupu v prípade programu Terraform vyzeral nasledovne:

```
resource "ec2_instance" "example" {
        count                = 10
        ami                  = "ami-v1"
        instance_type        = "t2.micro"
}
```

Pri takto jednoduchom príklade nie je rozdiel príliš veľký, líšia sa formy zápisu. Zjavný rozdiel však nastane pri akejkoľvek zmene v danej konfigurácii.

V praxi často nastane situácia, pri ktorej je nutné rozšíriť množstvo inštancií webových serverov. V prípade použitia procedurálneho prístupu je potrebná nová úloha, ktorá spustí počet inštancií, o ktoré chce správca zvýšiť aktuálny stav. V prípade deklaratívneho prístupu stačí, ak správca zmení výsledné množstvo inštancií a spustí daný nástroj. Podstatný rozdiel však nastáva pri zmene aktuálneho stavu, napr. sa môže jednať o nasadenie novej verzie webovej aplikácie. V takom prípade je pri použití procedurálneho prístupu nutné definovať nasledujúce kroky:

1. skript na ukončenie aktuálnych inštancií,
2. skript na nasadenie nových inštancií s novou verziou aplikácie,
3. spustenie daného programu.

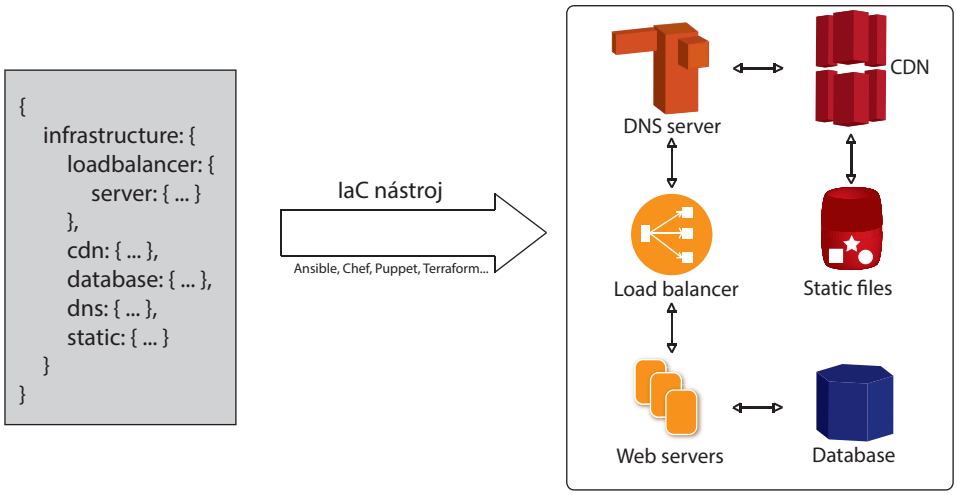
Deklaratívny prístup opäť zaručuje, že stačí pozmeniť pôvodný skript a spustiť program, ktorý sa postará o nastavenie. Nevýhodou tohto prístupu je strata kontroly nad vykonávanými krokmi pre dosiahnutie daného stavu.

2.2.2 Použitie nástroja

Samotné použitie zvoleného nástroja je pomerne jednoduchý proces, skladajúci sa z troch krokov:

- zostavenie požadovanej konfigurácie vo forme kódu pre zvolený nástroj,
- kontrola kódu (suchý štart) a ostré spustenie nástroja,
- overenie funkčnosti nasadenej infraštruktúry.

Možnú schému, ako si možno predstaviť tieto kroky, zjednodušene zobrazuje schéma na obr. 2.1. V tomto príklade je výstupom spracovanej konfigurácie pomocou IaC nástroja komplexná infraštruktúra zložená zo sieťových zariadení a aplikačných či dátových serverov.



Obr. 2.1: Schéma nasadenia IaC nástroja

2.3 Ansible

Automatizačný nástroj má využitie naprieč širokým spektrom zariadení, nezávisle na rozdelení zariadení [22]. Možno ho využiť ako na fyzické tak na virtualizované zariadenia, na koncové zariadenia, sieťové úložiská či sieťové zariadenia, ktoré podporujú zabezpečené pripojenie pomocou protokolu SSH. Podporované sú samozrejme všetky štandardne podporované OS, od Windowsu, cez MacOS po rôzne unixové a linuxové distribúcie.

Hlavnou výhodou tohto nástroja je práve široká podpora zariadení bez nutnosti inštalovať agenta na vzdialené zariadenie, či mať špecifické rozhranie pre dané zariadenie. Samotný manažment pomocou nástroja Ansible je pomerne jednoduchý aj pre začínajúcich správcov siete. Využíva procedurálny prístup konfigurácie, takže zmeny v konfigurácii vyžadujú väčší zásah, avšak za cenu plného prehľadu nad zmenami.

Podľa databázy portálu Datanyze, na základe ktorej zostavila rebríček najpoužívanejších automatizačných nástrojov [23], využíva tento nástroj približne 26,53% webových stránok z tejto databázy. Ďalšie uvedené nástroje ako Terraform od spoločnosti HashiCorp, či Puppet dosiahli trhový podiel na úrovni len približne 11,5%.

2.3.1 Moduly

Ansible modul je samostatná jednotka kódu, ktorá dokáže vykonávať definované akcie, môže byť volaná zo scenára alebo priamo z príkazového riadku. Úlohu pre modul definujú vstupné parametre, ktoré je možné danému modulu zavolať tromi spôsobmi v závislosti od volania modulu[24]. V prípade, že je modul volaný z príkazového riadku, sú jednotlivé parametre vkladané za volaný modul pomocou skratky:

```
ansible webservers -m command -a "/sbin/reboot -t now".
```

Takto definovaný príkaz popisuje reštart zariadení priradených do skupiny webservers pomocou príkazu `/sbin/reboot -t now`. Tento istý príkaz možno zapísať aj do scenára, vo forme YAML súboru:

```
- name: reboot the servers
  action: command /sbin/reboot -t now

- name: reboot the servers
  command: /sbin/reboot -t now
```

Ako je možné vidieť na ukážke vyššie, zvyčajne je na splnenie úlohy možné využiť nie len jeden ale viacero vhodných modulov. Takéto rozdielne zápisy budú líšiť okrem použitého modulu aj parametrami, špecifickejší modul bude vyžadovať menej vstupných parametrov pre vykonanie úlohy, naopak univerzálnejší ich bude potrebovať väčší počet.

Zápis do scenára má ešte jednu formu, ktorá sa nápadne blíži deklaratívnemu zápisu:

```
- name: restart webserver
  service:
    name: httpd
    state: restarted
```

V takomto zápise je definovaný výsledný stav infraštruktúry namiesto akcií, ktoré sa majú vykonať. Okrem parametrov, ktoré modul prijíma a následne na základe nich vykonáva akcie majú moduly aj možnosť spúšťať udalosti, pri ktorých sú spúšťané obslužné funkcie.

Ansible momentálne obsahuje rádovo stovky oficiálnych modulov [25], rozdelených do 21 kategórii, ako sú napríklad súborové, sieťové, systémové moduly či priamo moduly určené pre OS Windows. Medzi súborové moduly patria okrem iných aj moduly ako sú archive určený na komprimáciu súborov, či modul `read_csv` určený na čítanie súborov typu CSV. Pre OS Windows sú to napríklad moduly `win_defrag` určený na defragmentáciu disku či `win_regedit`, ktorý vykonáva určené akcie nad Windows registrami.

Samotné moduly možno rozdeliť na:

- oficiálne Ansible moduly, ktoré spravuje a vyvíja Ansible vývojový tím,
- certifikované moduly, ktoré spravujú tzv. Ansible partneri, teda známe firmy ako Cisco, NetApp či Nokia,
- moduly spravované komunitou, väčšina dostupných modulov patrí práve do tejto kategórie, do týchto modulov je možné prispieť podporou pri vývoji, či vývojom a zdieľaním vlastného modulu.

YAML

Serializačný štandard pre zápis dát v textovom formáte jednoducho čitateľný pre človeka [26]. Jedná sa o abstraktnejší formát ako formát JSON, nevyžaduje explicitné oddeľovanie pomocou množinových zátvoriek (angl. curly brackets). Rovnako ako JSON však využíva odsadenie, to je však možné len pomocou medzier, nie tabulátora, vzhľadom na chýbajúceho množinové zátvorky.

Základnými prvkami formátu YAML sú záznamy vo forme zoznamu, začínajúce na spojovník `-` a asociatívna dvojica, tzv. kľúč a hodnota, ktoré sú oddelené dvojbodkou [27]. V prípade kompaktnejšieho zápisu možno viacero záznamov zapísať v jednom riadku a umiestniť do množinových zátvoriek.

Jedná sa o štandardný formát podporovaný množstvom programovacích jazykov, ktoré obsahujú štandardné knižnice pre spracovanie YAML formátu. Momentálne sa nachádza v 3. verzii, oficiálne číslo verzie je 1.2.

Nasledujúca ukážka zobrazuje príklad záznamu zamestnanca vo formáte YAML, v kompaktnej a klasickej forme [28]:

```
- {name: John Smith, age: 33, skills: C, C++, python, OOP}

- name: Mary Smith
  age: 27
  skill:
    - C
    - C++
    - SQL
```

Tieto zápisy sú plne kompatibilné, druhý spôsob sa používa vo väčšej miere najmä kvôli prehľadnosti. Prvý, kompaktnejší zápis, je používaný najmä pri strojovom spracovaní komplexných konfiguračných súborov, podobne ako napr. minimalizované súbory kaskádových štýlov (CSS) pri webových stránkach.

Jinja

Jednoduchý textový šablónový formát umožňujúci generovať ľubovoľný výstupný textový formát (napr. HTML, CSV, JSON a iné.) [29]. Šablóna obsahuje premenné a výrazy, ktoré sú nahradené pri generovaní výstupného formátu.

Premenná, ktorú možno nájsť v šablóne ako `{{ ... }}`, je pri generovaní jednoducho nahradená hodnotou, priradenou k danej premennej. Tento spôsob je využitý v konfigurácii nástroja Ansible, prostredníctvom formátu Jinja sú do častí scenárov vkladané premenné. V prípade, že sa jedná o zložitú premennú, možno k jednotlivým častiam pristupovať pomocou tzv. bodkového zápisu (`.`) alebo pomocou tzv. "__getitem__" zápisu známeho z jazyka python (`[' ']`). Príklady na oba zmienené spôsoby prístupu k zložitým premenných zobrazuje nasledujúci výpis:

```
{{ foo.bar }}
{{ foo['bar'] }}
```

Výraz, zapísaný v šablóne ako `{% ... %}`, umožňuje zápis komplexných príkazov v jazyku Python. Nasledujúci príklad pri generovaní vytvorí HTML zápis obsahujúci neusporiadaný zoznam obsahujúci tri odkazy na uvedené podstránky pomocou cyklu for, v každej iterácii sú premenné `href` a `caption` nahradené dvojicou zo zoznamu.

```
<ul>
{%
  for href, caption in [('index.html', 'Index'),
  ('about.html', 'About'), ('downloads.html', 'Downloads')]
%}
<li><a href="{{ href }}">{{ caption }}</a></li>
{% endfor %}
</ul>
```

2.3.2 Scenáre

V Ansible terminológii označované ako `Playbooks`, sú spôsobom manažmentu konfigurácie [30]. Scenáre sú základom pre rôzne situácie, od úplne jednoduchých scenárov, cez nasadenie viacerých zariadení až po komplexné siete. Okrem manažmentu konfigurácie umožňujú aj orchestráciu manuálne definovaných krokov, ktoré môžu byť vrátené v prípade potreby. Výhodou najmä v prípade komplexnejších scenárov je možnosť synchrónneho aj asynchrónneho spúšťania. V prípade asynchrónneho spúšťania konfigurácie Ansible nečaká na dokončenie úloh na predošlom zariadení, ale spúšťa ich sekvenčne za sebou a následne asynchrónne prijíma výsledky a zobrazuje ich správcovi.

Každý scenár sa skladá z jedného alebo viacerých častí, nazvaných `plays`. Každá táto časť obsahuje skupinu zariadení, na ktorých má byť vykonaná a zoznam úloh, ktoré majú byť vykonané na týchto zariadeniach. Nasledujúci príklad názorne ilustruje scenár zložený z dvoch častí, pričom každá časť vykonáva úlohy na inej skupine zariadení.

```
---
- hosts: webservers
remote_user: root

tasks:
- name: ensure apache is at the latest version
  yum:
    name: httpd
    state: latest
```

```
- name: write the apache config file
  template:
    src: /srv/httpd.j2
    dest: /etc/httpd.conf

- hosts: databases
  remote_user: root

tasks:
- name: ensure postgresql is at the latest version
  yum:
    name: postgresql
    state: latest
- name: ensure that postgresql is started
  service:
    name: postgresql
    state: started
```

Pri komplexnejších scenároch je často využívané vkladanie súborov, ktoré obsahujú jednotlivé časti súborov [31]. V závislosti od spôsobu kontroly vkladaného súboru rozlišuje Ansible dva spôsoby, akými je možné vkladať súbory do scenárov. Pri použití príkazu `import_*` (statické vkladanie), kde hviezdička reprezentuje druhú časť príkazu, napr. `playbook`, `task` a pod., je vkladaný súbor kontrolovaný už pri preklade, v prípade chyby Ansible nedovolí spustenie scenára. Použitím príkazu `include_*` táto kontrola nenastáva (dynamické vkladanie), teda v prípade problému ukončí Ansible vykonávanie scenára a vyhlási chybu. Scenáre resp. časti scenárov sú vkladané pomocou príkazu `import-playbook`, pri vkladaní úloh či iných typov súborov je obvykle možno použiť oba zmienené spôsoby.

Úlohy

V dokumentácii označované výrazom `Tasks` sú dielčie časti, ktoré volajú moduly spolu so zadanými parametrami. Parametre závisia od zadaného modulu, potrebné je definovať parametre, pri ktorých správcovi nevyhovujú predvolené hodnoty. Príklad jednoduchej úlohy zobrazuje detail úlohy nižšie:

```
- name: ensure apache is at the latest version
  yum:
    name: httpd
    state: latest
```

Na výpise možno vidieť štandardnú úlohu obsahujúcu názov, ktorý Ansible používa pri výpisoch (v prípade splnenia úlohy či chyby), použitý modul a jeho parametre. Parameter názov nie je nevyhnutný, no je odporúčaný pre lepšiu orientáciu

správcu vo výpise. V tomto konkrétnom prípade možno vidieť, že sa jedná o úlohu, ktorá má pomocou modulu `yum` (obsluhujúci rovnomenného správcu balíčkov), zabezpečiť najnovšiu verziu balíka httpd. Správca nemusí riešiť, či je daný balíček nainštalovaný alebo akú má aktuálnu verziu, o to sa postará Ansible.

V množstve prípadov je vhodné použiť premenné, najmä v prípade vykonávania jednej úlohy na viacerých zariadeniach. V takom prípade je možné definovať zoznam parametrov, ktoré sú do Ansible vložené pomocou šablónovacieho formátu Jinja, bližšie spomenutý v časti 2.3.1. Nasledujúci príklad zobrazuje vykonanie úlohy pre zariadenie, ktorého názov je uložený v premennej `vhost`:

```
tasks:
- name: create a virtual host file for {{ vhost }}
    template:
    src: somefile.j2
    dest: /etc/httpd/conf.d/{{ vhost }}
```

Roly

Roly (v oficiálnej terminológii nazývané `roles`) sú spôsobom skladania scenára z definovanej súborovej štruktúry [32]. Každá rola musí obsahovať minimálne jeden definovaný priečinok, ďalšie priečinky sú voliteľné v prípade potreby, avšak nie nutné. Obvykle existuje spoločná konfigurácia, uložená v súborovej štruktúre s názvom `common`, ktorá definuje informácie spoločné pre všetky alebo väčšinu definovaných rolí. V prípade, ak má rola inak definovanú konfiguráciu ako spoločná konfigurácia, je daná časť spoločnej konfigurácie ignorovaná. Každý priečinok v súborovej štruktúre musí povinne obsahovať súbor `main.yaml`, ďalšie súbory sú voliteľné (obsahujúce napr. úlohy závislé na cieľovej platforme).

Základnými priečinkami v súborovej štruktúre roly sú:

- `tasks` – úlohy, ktoré majú byť v rámci danej role vykonané
- `handlers` – obsluhy, ktoré reagujú na signály z úloh
- `defaults` – preddefinované premenné pre rolu
- `vars` – ďalšie premenné pre danú rolu
- `files` – súborov nasadzované v rámci role
- `templates` – šablóny nasadzované v rámci role
- `meta` – meta informácie pre danú rolu

Nasledujúci príklad ilustruje možnú súborovú štruktúru: v koreňovom priečinku sa nachádzajú súbory obsahujúce jadro scenára, teda definíciu skupiny definovaných zariadení vrátane ďalších parametrov. Primárne sa jedná o zaradenie do roly, doplňujúce informácie k danej role pre danú skupinu zariadení a pod. Okrem týchto

súborov sa v koreňovom priečinku nachádza zložka `roles`, ktorá už obsahuje súborové štruktúry pre definované roly, vrátane spoločnej súborovej štruktúry `common`, ak sa používa.

```
site.yml
webservers.yml
fooservers.yml
roles/
  common/
    tasks/
    handlers/
    files/
    templates/
    vars/
    defaults/
    meta/
webservers/
  tasks/
  defaults/
  meta/
```

Súbory v koreňovom priečinku popisujúce skupinu zariadení a ich priradené roly môže vyzerať podobne ako zobrazuje nasledujúci výpis. Skupina zariadení pod názvom `webservers` patrí do rovnomennej role, pričom jej ako parametre definuje dostupný port, pod ktorým bude dostupná webová služba a koreňový priečinok webovej aplikácie.

```
- hosts: webservers
  tasks:
  - include_role:
    name: foo_app_instance
  vars:
    dir: '/opt/a'
    app_port: 5000
```

Obsluhy

Obsluhy, v originálnej terminológií uvádzané ako `Handlers` sú špecifickým druhom úloh, ktoré sú vykonávané len pri zaslaní signálu na konci vykonávania úlohy [30]. Hlavným špecifikom týchto úloh je ich jednorazové vykonanie, bez ohľadu na to, koľkokrát dostanú signál pomocou parametra `notify` v úlohe. Typické použitie obsluhy môže byť reštart služby po zmene jej konfiguračných súborov, takéto použitie ilustruje výpis nižšie.

```
- name: template configuration file
  template:
```

```
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - restart apache
```

Po zmene konfiguračného súboru `/etc/foo.conf` pomocou šablóny `template.j2` sú zavolané obsluhy s názvami `restart memcached` a `restart apache`. Zápis týchto dvoch obslužných funkcií zobrazuje výpis nižšie. Alternatívou k týmto obsluhám by bolo vloženie týchto úloh priamo do úloh, v niektorých prípadoch to však nemusí byť vhodné. Použitie obslúh navyše zvyšuje prehľadnosť vykonávaných krokov v prípade problémov.

```
handlers:
  - name: restart apache
    service:
      name: apache
      state: restarted
```

2.3.3 Trezor

Z anglického originálu `Vault`, je v Ansible spôsob šifrovaného ukladania citlivých informácií [33]. Citlivé informácie možno rozumieť ako premenné napr. heslá a iné prístupové údaje, konfiguračné súbory a pod. Pracuje v dvoch režimoch, tým prvým je šifrovanie na úrovni premenných, druhým spôsobom je šifrovanie na súborovej úrovni. Pre zvýšenie bezpečnosti možno okrem samotných súborov obsahujúce citlivé údaje šifrovať aj súbory rolí, obslúh či úloh, ktoré obsahujú názvy premenných pristupujúcich k citlivým údajom.

Každý trezor má svoj unikátny identifikátor (ID) a jedno alebo viacero hesiel pre prístup k šifrovaným údajom. Pri práci s viacerými trezormi je možný výber aktuálneho trezoru pomocou parametra `--vault-id`, ktorý vyžaduje ID trezora a samotné heslo (dané cestou k súboru obsahujúce heslo alebo vložením hesla pomocou terminálu). V prípade použitia hesla je využívaný zápis: <id trezora>@<súbor s heslom>, napr. pre trezor s ID `staging` a heslom uloženým v súbore `staging_passwd` bude zápis vyzerať nasledovne: `staging@staging_passwd`.

Pre účely šifrovania Ansible využíva štandard AES v režime čítača (CTR) s dĺžkou kľúča 256 bitov, je možné ho zmeniť. Spôsob šifrovania je zaznamenaný pre daný súbor či premennú. Nasledujúci príklad ilustruje zápis terminálového príkazu na šifrovanie premennej `the_secret` využitím súboru `foobar` ako zdroja hesla.

```
ansible-vault encrypt_string --vault-password-file a_password_file
↪  'foobar' --name 'the_secret'
```

Výsledok operácie je následne vypísaný do terminálu. Symbol rúry (|) označuje zalomenie riadku, signalizuje pokračovanie príkazu/úlohy v ďalšom riadku.

```
the_secret: !vault |
        $ANSIBLE_VAULT;1.1;AES256
        623133653966623430613934643361633837643737646136336536
        343062313864336264366233616134333665353966363534333632
        666535333761666131620a663537646436643839616531643561 63
        396265333966386166373632626539326166353965363262633030
        333630313338646335303630343862666666137650a3536386434
        356666336339643663386330666232346164323732313333316564
```

Šifrovanie na úrovni súborov ponúka, okrem klasického šifrovania, úpravy šifrovaného súboru či dešifrovania, aj možnosť tzv. `prekľúčovania` (angl. `rekeying`), teda zmeny hesla, ktorá sa skladá z dešifrovania šifrovaného súboru, zašifrovaním súboru novým heslom a jeho uložením do trezora. Nasledujúci príklad zobrazuje šifrovanie nešifrovaného súboru a následnú zmenu hesla.

```
ansible-vault create --vault-id staging@devel_passwd foo.yml
ansible-vault rekey --vault-id staging@devel_passwd --new-vault-id
↪  staging@staging_passwd foo.yml
```

V príklade vidieť, že je použitý trezor s identifikátorom `staging`, pri šifrovaní nešifrovaného súboru je do tohto trezora uložené heslo `devel_passswd`, pri následnej zmene hesla je obsah trezora zmenený a je v ňom uložené heslo `staging_passwd`.

Spustenie šifrovaného súboru

Pri spúšťaní šifrovaného yaml súboru, príp. iného súboru patriaceho do súborového systému scenára je potrebné pri spúšťaní programu Ansible pridať parameter `--ask-vault-pass`, ktorý umožní zadať heslo pomocou konzoly. V prípade využitia súboru s heslom sa používa parameter spúšťania `--vault-pass-file`. V prípade, že chce správca spustiť ansible scenár s hlavným súborom `main.yml`, obsahujúci šifrovaný súbor pomocou ansible vault, bude ním použitý príkaz vyzerať napríklad takto:

```
ansible-playbook main.yml --vault-id staging --ask-vault-pass
```

po zadaní tohto príkazu bude správca vyzvaný k zadaniu hesla a po jeho úspešnom zadaní vykoná ansible príkazy zadané v konfigurácii.

3 Štruktúra generátora konfigurácie

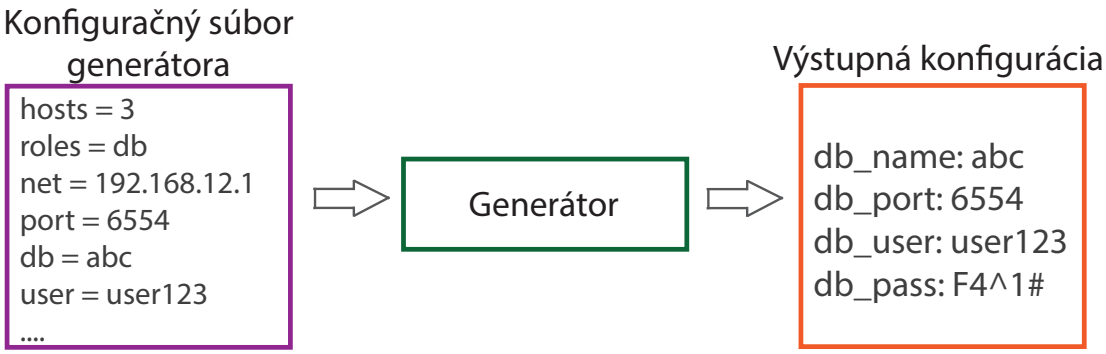
Praktickou časťou práce je generátor konfigurácie pre program Ansible a shell skript, ktorý následne konfiguráciu overuje použitím tzv. suchého behu (angl. `dry run`).

3.1 Režimy generátora

Samotný generátor dokáže pracovať v editačnom režime a v režime plného generátora. V editačnom režime program pracuje s už existujúcou konfiguráciou, ktorú podľa zadaných parametrov upravuje. Režim plného generátora dokáže generátor vygenerovať celú konfiguráciu, teda súborovú štruktúru, podľa zadaných parametrov. V prípade práce s existujúcou konfiguráciou program overuje, či je konfigurácia v otvorenom režime alebo je šifrovaná pomocou Ansible-Vault, v prípade šifrovania je vyžiadaný symetrický šifrovací kľúč a po úprave je konfigurácia znova zašifrovaná.

3.1.1 Režim plného generátora

V tomto režime je generátor schopný vygenerovať kompletnú Ansible konfiguráciu zo zadaného konfiguračného súboru vo formáte vhodnom pre takýto zápis. Generátor má definované kategórie akcií, podľa ktorých zaradí jednotlivé nastavenia do správnej sekcie, napr. premenné uloží do jedného výstupného súboru a pod. Vizualizáciu tohto režimu zobrazuje obr. 3.1. Pre zložitosť zadávania jednotlivých parametrov, resp. pre ich lepšiu prehľadnosť je pre tento režim vhodnejší spôsob zadávania parametrov pomocou konfiguračného súboru, pričom pri spúšťaní je generátoru poskytnutý režim a cesta k tomuto súboru.



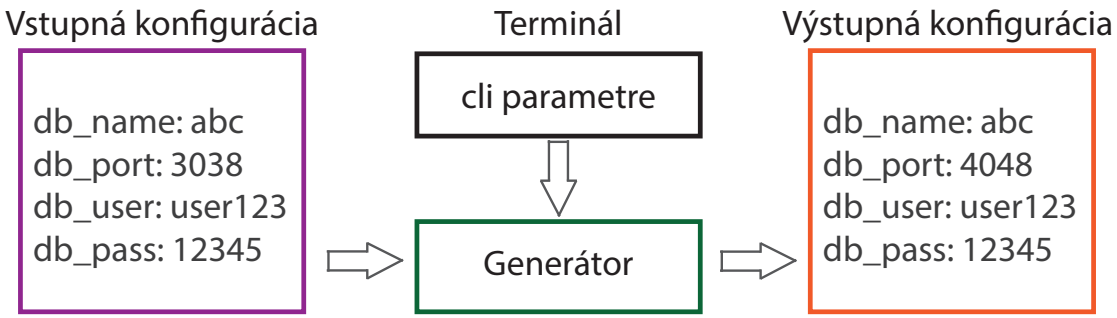
Obr. 3.1: Ilustrácia režimu Generátora.

3.1.2 Editačný režim

V editačnom režime program podporuje vyhľadávanie a nahrádzanie hodnôt v súborovej štruktúre Ansible konfigurácie podľa zadaných parametrov. V závislosti od typu nahradených hodnôt môže byť nahrádzanie vygenerovanou hodnotou zo zadaného rozsahu alebo náhodným výberom zo zoznamu definovaných hodnôt. V prípade potreby je možné využiť aj režim statickej úpravy, v ktorej je pri spustení tohto programu uvedený názov kľúča a nová hodnota pre tento kľúč.

Generovanie hodnôt z rozsahu

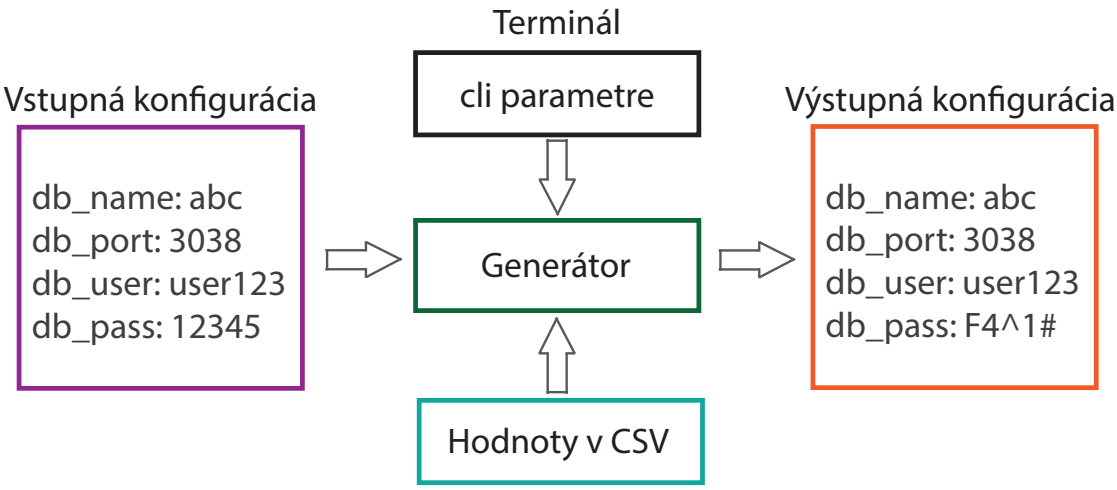
Možnosť generovania hodnôt z rozsahu hodnôt je využitý pri úprave IP adries a portov v konfigurácii, ktoré sú zadávané priamo ako parameter programu. V prípade potreby je aj tieto vstupy možné zadávať staticky, program tieto hodnoty overuje a v prípade problémov informuje o nesprávnom vstupe. Vizualizáciu tohto režimu zobrazuje obr. 3.2.



Obr. 3.2: Ilustrácia editačného režimu.

Náhodný výber hodnôt

V prípade prihlasovacích mien, hesiel a iných údajov uložených vo forme textových reťazcov program obsahuje náhodný výber zo zoznamu týchto reťazcov. Výber spočíva v určení počtu možných reťazcov (hodnôt) zo zadaného textového súboru vo formáte `CSV` (angl. `comma separated values`), v ktorom sú jednotlivé záznamy oddelené čiarkou. Následne je pomocou pseudonáhodnej funkcie `random()` vygenerovaný index reťazca a ten je vybraný pre zápis do cieľovej konfigurácie. Jednoduchú schému fungovania zobrazuje obr. 3.3.



Obr. 3.3: Ilustrácia editačného režimu so súborom CSV.

3.2 Návrh generátora

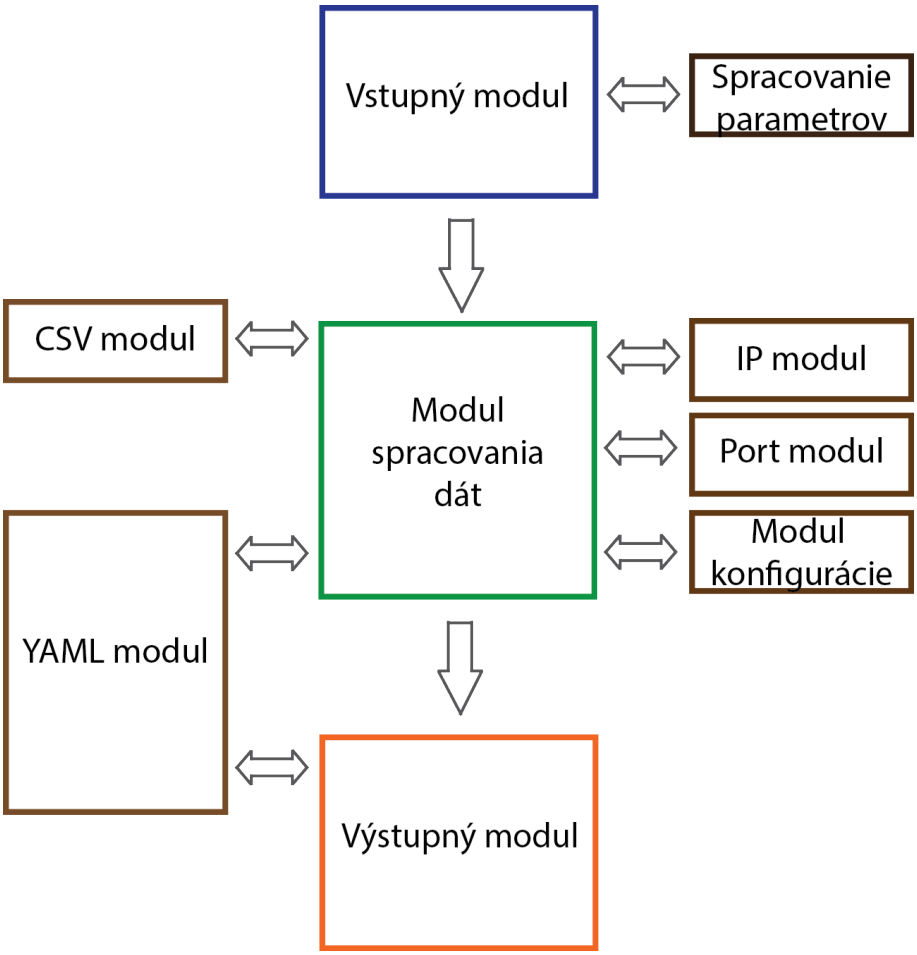
Teoretický návrh jednotlivých modulov generátora vychádza z požiadaviek na jeho funkčnosť. Pre jednoduchú rozšíriteľnosť programu v neskoršej fáze bol uprednostnený modulárny návrh programu pred jednoliatym celkom, čo v konečnom dôsledku zvýši aj prehľadnosť programového kódu. Navrhnutú štruktúru programu možno rozdeliť na časť načítania vstupných dát, spracovanie dát a výstupnú časť, ktorej výsledok je samotná konfigurácia. Schému prepojenia jednotlivých modulov zobrazuje obr. 3.4. Vstupné informácie sú prijaté do vstupného modulu a spracovanie pokračuje naprieč schému smerom nadol. Jednotlivé moduly a ich prepojenia sú popísané v nasledujúcich kapitolách.

3.2.1 Vstupný modul

Vstupné údaje je možné riešiť viacerými spôsobmi, medzi najviac používané spôsoby patria zadávania parametrov:

- do formulára v grafickom režime programu,
- ako argumenty spúšťania programu v termináli,
- alebo do konfiguračného textového súboru.

Každé z uvedených riešení má svoje výhody ako aj nevýhody. Grafický režim bol vyhodnotený ako vhodný pre menej odborných používateľov. Vhodnejšie bude pre toto použitie využitie textového zadávania parametrov. Pre rýchlejšiu implementáciu bol zvolený režim pomocou parametrov terminálu, ktorý je vyhodnocovaný pomocou štandardného programu na vyhodnocovanie argumentov `get-opt`. Na základe



Obr. 3.4: Schéma vnútornej štruktúry generátora.

zadaného režimu v tejto časti prebieha kontrola správnosti zadaných parametrov a v prípade chýbajúcich údajov na to program upozorní. Pre informáciu o jednotlivých parametroch obsahuje program aj argument help, ktorý vypíše všetky povinné parametre pre dostupné režimy.

3.2.2 Modul spracovanie dát

Po spracovaní vstupných parametrov nasleduje postupnosť funkcií závislá na vybranom režime generátora. V prípade editačného režimu je to otvorenie zdrojových YAML súborov a načítanie existujúcej konfigurácie, ktorá je následne upravovaná. Úpravy sú opäť závislé na zadaných parametroch, jedná sa o statické nahradenie zadaného parametra či generovaním hodnoty z predom daného rozsahu v termináli alebo zoznamu uloženého v textovom súbore. V prípade režimu plného generátora nastáva v tomto kroku pomocou modulu konfigurácie vytvorenie súborovej štruktúry potrebnej pre uloženie konfigurácie a vygenerovanie jednotlivých konfigurácií

v slovníkovej forme, ktorá je uložená v pamäti programu.

Pomocné moduly

Pre spracovanie konfigurácie sú v programe použité pomocné moduly, v schéme označené odtieňmi hnedej. Tieto moduly umožňujú overovať zadané statické parametre v prípade, že sa jedná o IP adresu alebo port, alebo tieto parametre náhodne vygenerovať zo zadanej siete resp. rozsahu. Pre náhodný výber zo zoznamu hodnôt vo formáte CSV je použitý pomocný modul na čítanie súborov v tomto formáte. Pre vytváranie súborovej štruktúry pre konfiguráciu a prechádzanie medzi súbormi danej konfigurácie je v module spracovania dát využitý tzv. modul konfigurácie.

3.2.3 Výstupný modul

Po úprave alebo vygenerovaní konfigurácie nasleduje jej serializácia, teda zápis do textovej podoby na disk. V prípade konfigurácie pre Ansible je to už spomenutý formát YAML, pričom jednotlivé úlohy sú chronologicky usporiadané a logicky štruktúrované tak, aby sa v prípade konfigurácie pre viaceré zariadenia neopakovali pokyny platné pre viaceré zariadenia či skupiny zariadení. Rozhranie modulu yaml je navrhnuté tak, aby v prípade potreby výstupu do iného formátu bolo možné jednoducho napísať rozhranie aj pre nový formát a jednoducho zmeniť výstupný formát.

4 Implementácia nástroja

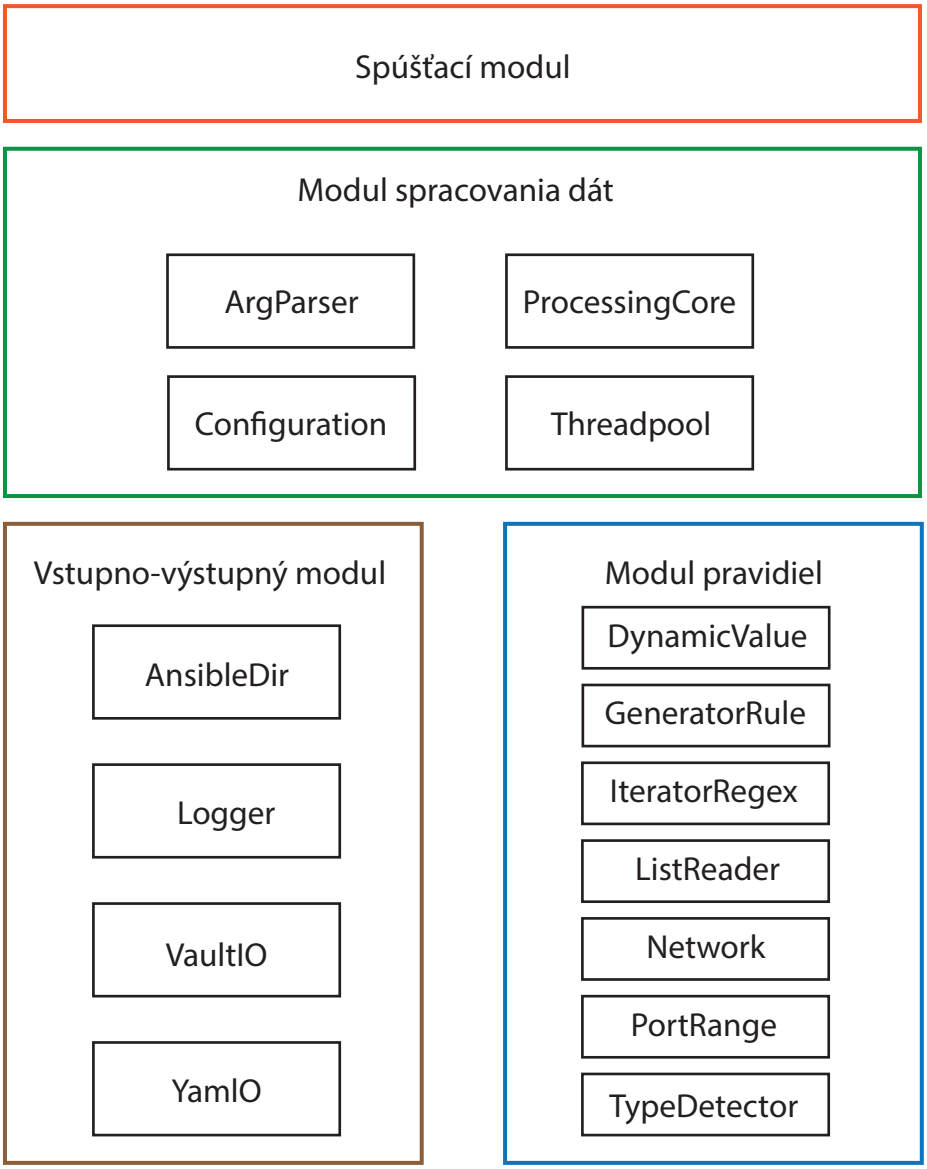
Pre praktickú implementáciu bol v zadaní určený jazyk Python. Jedná sa o programovací jazyk vhodný na tento účel, kedže aj samotný nástroj Ansible je vyvinutý v tomto jazyku. Navyše aj textový formát konfigurácie, formát YAML, rovnako vychádza z typov definovaných v tomto jazyku. Samotná implementácia nástroja vychádza z čistého jazyka Python a systémových modulov, bez použitia nutnosti využívania externých knižníc. V prípadoch, ktoré pracujú so šifrovanou konfiguráciou, je možné externými knižnicami urýchliť kryptografické operácie pri čítaní a zápise šifrovanej konfigurácie.

4.1 Vývoj modulov

Každý modul v generátore je reprezentovaný vlastnou triedou, pričom hlavné moduly sú volané zo spúšťacieho súboru `ans_gen.py`. Následne už moduly komunikujú a volajú potrebné funkcie z pomocných modulov tak, aby zabezpečili požadovanú funkčnosť. Moduly využívajú najmä návrhové vzory `singleton` a `fasáda`. Vzor singleton je použitý pri moduloch, ktoré nevyžadujú ukladanie vnútorného stavu a jednotlivé funkcie spolu logicky súvisia, sú však nezávislé od seba. Tento vzor je použitý napr. pri module IP adries alebo portov, v ktorých poskytujú funkcie na generovanie hodnoty z vopred definovaného rozsahu alebo na overenie správne zadanej hodnoty. Návrhový vzor fasáda je napríklad použitý v module Yaml IO, v ktorom poskytuje zjednodušené rozhranie systémového modulu `yaml` vhodné pre použitie v generátore. Schému jednotlivých modulov zobrazuje obr. 4.1. Detailnú adresárovú štruktúru obsahujúcu jednotlivé moduly, vrátane jednotlivých častí (submodulov) zobrazuje príloha A.

4.1.1 Modul spracovania dát

Modul zabezpečujúci samotnú náhodnosť dát vo vygenerovanej konfigurácii. Tento celok pozostáva zo 4 častí. Prvou časťou je spracovanie argumentov z terminálu, kontroluje ich správnosť a vytvára interné programové nastavenia pre beh nástroja. Ďalšími časťami modulu sú `Configuration`, ktorá poskytuje programovú abstrakciu nad yaml súborom pre čítanie a zápis do súboru a `Threadpool`, ktorý sa používa na paralelizáciu v režime generátora. Poslednou a najdôležitejšou časťou modulu spracovania dát je jadro modulu spracovania dát, ktorá obsahuje hlavnú programovú logiku oboch režimov.



Obr. 4.1: Architektúra modulov implementovaného generátora.

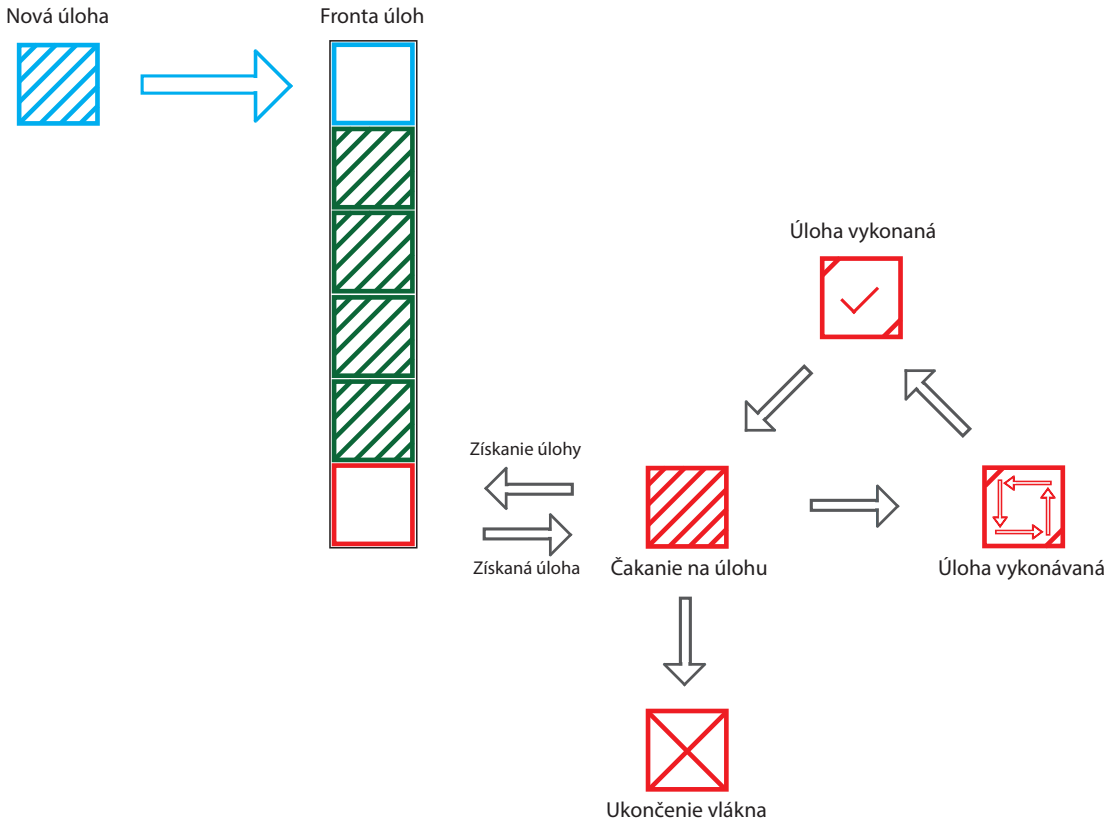
Modul Threadpool

Súčasť generátora zabezpečujúca paralelný beh generovania viacerých konfigurácií. Voľne je tento názov možné preložiť ako skupinu vlákien, ako celok má však svoje špecifiká, v porovnaní so samostatným vláknom. Na rozdiel od samostatného vlákna, ktorého životnosť je počas vykonávania jednej úlohy, životnosť vlákna v Threadpoole je obvykle podmienené určitým signálom alebo príznakom.

V prípade tohto generátora je to vyprázdnenie fronty úlohy. Tá je vopred naplnená tak, že pre každú iteráciu generovania je vytvorená úloha, ktorá je následne vložená do fronty (modrý priebeh). Každá úloha (zelené časti schémy) je samostatný

celok, ktorý keďže nemá závislosti ani nadväznosť na ďalších úlohách vo fronte, je možné vykonávať paralelne. Samotná úloha obsahuje všetky potrebné operácie pre jednu iteráciu, obsahuje kópiu pravidiel, pomocou ktorých generuje výstupné hodnoty a cestu k pracovnému priečinku, v ktorom upravuje konfiguráciu.

Po spustení ThreadPoolu je vytvorené definované množstvo vlákien, ktoré majú vykonávať úlohy z danej fronty. Každé vlákno si na začiatku vyžiada úlohu z fronty úloh, vykonáva úlohu a po jej dokončení hlási úspešné splnenie úlohy a opakuje získavanie úlohy. Po dokončení všetkých úloh je každé vlákno v Threadpoole ukončené (červený priebeh). Graficky tento postup znázorňuje nasledujúca schéma č. 4.2.



Obr. 4.2: Implementácia modulu Threadpool v režime generátora.

4.1.2 Vstupno-výstupný (IO) modul

Zabezpečuje vstupné a výstupné operácie, napr. čítanie a zápis do súboru na disk či komunikáciu so štandardným vstupom (`stdin`) alebo výstupom (`stdout`) a pod. Pre potreby generátora sú to najmä čítanie a zápis (do) yaml súborov v otvorenej forme (angl. `plain text`) alebo šifrovanej forme. Tieto operácie pre vyššie programové moduly zabezpečujú submoduly Yaml IO a Vault IO, ktorý je postavený na knižnici

`ansible_vault`. Okrem týchto operácií zabezpečuje zaznamenávanie (tzv. logovanie) udalostí na určený výstup, túto funkcionalitu zabezpečuje submodul `Logger`. Prednastaveným výstupom pre logovanie je štandardný výstup operačného systému (`stdout`). V neposlednom rade tento modul obsahuje submodul `AnsibleDir`, slúžiaci pre vytváranie cieľovej cesty a kopírovanie šablónovej konfigurácie na cieľové umiestnenie.

Modul ansible_vault

Python`wrapper` [1] umožňujúci využívať fukcie nástroja Ansible Vault. Modul je bližšie popísaný v kapitole 2.3.3). Pomocou funkcií tohto modulu prebieha dešifrovanie vstupnej šablóny konfigurácie a konfiguračného súboru generátora a šifrovanie vygenerovaných konfigurácií, ako možno vidieť na obr. 4.3. Využitím tohto modulu vznikajú tri kombinácie šalblónových vstupov a generovaných výstupov:

- nešifrovaná šablóna + prázdne heslo => nešifrovaný generovaný výstup,
- nešifrovaná šablóna + zadané heslo => generovaný výstup čiastočne šifrovaný,
- šifrovaná šablóna + zadané heslo => výstup šifrovaný heslom vstupnej šablóny.

Pri šifrovaní výstupnej šablóny vygenerovanej z nešifrovanej šablóny je šifrovaná len tzv. citlivá časť súborov, teda súbory do ktorých počas generovania prebiehal zápis hodnôt. Kombinácia šifrovanej šablóny a nezadaného hesla je považovaná za neplatnú a generátor v tomto prípade vypíše varovnú hlášku a ukončí svoju činnosť.

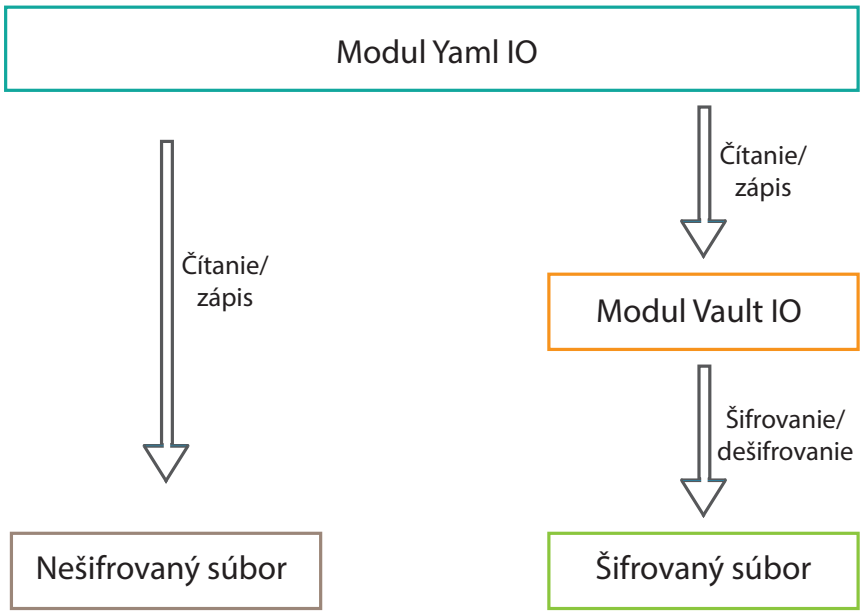
4.1.3 Modul pravidiel

Podporný modul pre modul spracovania dát. Obsahuje abstrakciu pre jednotlivé pravidlá a nízkoúrovňové funkcie pre ich spracovanie, ako napr. detekcia typu pravidla, spoločné rozhranie pre jednotlivé pravidlá alebo samotné generovanie náhodnej hodnoty podľa definície pre zadané pravidlo.

Pravidlo

Samotné pravidlo je možné definovať ako objektovú reprezentáciu požiadavky na náhodne generovanú hodnotu. Nástroj podporuje 2 základné typy pravidiel, statické a dynamické. Statické pravidlo možno definovať ako priamo definovanú výstupnú hodnotu, pre vygenerovanie výstupnej hodnoty nie je potrebné žiadne generovanie. Pri statických pravidlách je možné využiť iteračnú premennú `<#>`, ktorá bude na výstupe nahradená poradovým číslom iterácie (začína číslom 0).

[1] voľne možno preložiť ako tzv. obálka, kód umožňujúci využitie funkcionality v inom ako pôvodnom programovacom jazyku alebo kóde



Obr. 4.3: Schéma čítania súborov využitím modulu Vault IO.

Dynamické pravidlá naopak využívajú generátor náhodných čísel, výsledok ktorého je priamo alebo nepriamo využitý vo vygenerovanej hodnote. Dynamické pravidlá sa delia na tri základné typy:

- `Network` – generovanie IP adresy určenej pre koncové zariadenia zo zadanej adresy siete a prefixu,
- `PortRange` – generovanie portu zo zadaného rozsahu, kontroluje správnosť zadaného rozsahu,
- `ListGenerator` – generovanie hodnoty zo zadaného zoznamu možných výstupov vo forme textového súboru.

Iteračná premenná

Špeciálny reťazec znakov `<#>` určený pre variabilný číselný vstup. V praxi je výhodné použiť iteračnú premennú v prípade, ak má každá generovaná konfigurácia obsahovať inú číselnú hodnotu, napr. IP adresu z iného rozsahu a pod. Tento reťazec bude počas behu generátora nahradený indexom danej iterácie, začínajúcim číslom 0. Pre vyššiu mieru flexibility generátor podporuje základné matematické operátory (+, -, *, / a %) a jeden číselný operand, čím je možné dosiahnuť aj iné požadované výstupné hodnoty. Pomocou zápisu `<#+1>` je napríklad možné dosiahnúť počiatočný index s hodnotou 1 pre prvú vygenerovanú konfiguráciu. Typické príklady použitia uvádza nasledujúci výpis:

```
server_name: WebServer<#+100>
server_addr: 192.168.<#+10>.0
```

V prípade statického názvu je možné zvoliť takmer ľubovoľný počet iterácií, teda vygenerovaných konfigurácií. Jediné obmedzenie je v tomto prípade ohraničené maximálnou hodnotou dátového typu int[2] pre danú implementáciu jazyka Python pre používaný operačný systém.

Konfiguračný súbor

Každá sekcia začína komentárom popisujúcim danú sekciu (nepovinné) a následne obsahuje jednotlivé pravidlá. Všeobecne platí, že konfiguračný súbor má množinu povinných pravidiel, medzi ktoré sa radí počet iterácií a minimálne jedno pravidlo (statické alebo dynamické), v opačnom prípade je konfiguračný súbor považovaný za neplatný. Na poradí jednotlivých pravidiel v rámci jednej sekcie nezáleží.

Pravidlo `iterations` vyjadruje požadovaný počet vygenerovaných konfigurácii, `name` pomenúva danú konfiguráciu, ktorú generátor využíva len na informáciu používateľa a logovanie. Pravidlá obsahujúce cesty pre vstupnú šablónovú konfiguráciu `input` a cestu pre výstupné konfigurácie `output` majú nižšiu prioritu ako parametre v termináli (platí však, že oba parametre musia byť uvedené aspoň raz – v konfiguračnom súbore alebo ako argument programu v termináli)

Vzorový konfiguračný súbor generátora, obsahujúci všetky vyššie uvedené pravidlá, uvádza nasledujúci výpis:

```
# General settings
general:
- iterations: 30
  name: YAML generator
  input: ./app_test/input/lamp_simple_centos8/
  output: ./app_test/output/

# Static values
static:
- repository: "https://github.com/bennojoy/mywebapp.git"

# Dynamic values
dynamic:
- ntpserver: 192.168.<#>.0/26
  httpd_port: 2042-5142
  upassword: ./include/test/source/passwords.txt
```

[2]Maximálnu hodnotu možno zobraziť v Python konzole príkazom sys.maxsize, typicky je to hodnota (2^63)-1.

Po spracovaní tohto konfiguračného súboru generátorom bude v koreňovom priečinku programu vytvorená štruktúra `app_test/output/`, ktorý bude obsahovať priečinok označený časom a dátumom spustenia generátora, ten už bude obsahovať priečinok pre každú iteráciu s vygenerovanou konfiguráciou. Príklad výstupnej konfigurácie pre prvú iteráciou (index iterácie 0) zobrazuje nasledujúci výpis:

```
ntpserver: 192.168.0.5/26
httpd_port: 3122
upassword: TOP_SECRET_PASSWORD_GENERATED_FROM_PASSWORD_FILE
repository: "https://github.com/bennojoy/mywebapp.git"
```

Výstupná konfigurácia môže pozostávať z jedného alebo viacerých súborov obsahujúcich citlivé údaje, generátor vyhľadá a následne nahradí jednotlivé hodnoty vo všetkých súboroch, v ktorých sa dané kľúče vyskytujú.

5 Testovanie

Testovanie implementácie možno rozdeliť na tri samostatné kroky. Počas implementácie prebiehalo testovanie generátora, realizované formou unittestov (testovanie jednotlivých metód a modulov) a testovanie aplikácie (testovanie správania generátora na rôzne kombinácie validných a nevalidných vstupov). Tretím krokom bolo bezpečnostné testovanie vykonané formou statickej analýzy špecializovaným nástrojom.

5.1 Testovanie generátora

Testovanie zamerané na kvalitu vyvíjaného generátora už počas samotnej implementácie jednotlivých modulov, pričom vývoj nasledujúcich modulov je podmienený splnením podmienok – úspešným vykonaním testov. Tento proces sa v anglickej terminológii označuje ako `Test-driven development`[34], čo možno preložiť ako vývoj založený na testoch. Pre tento účel slúžia tzv. jednotkové testy (angl. `unittests`), ktoré testujú funkcionalitu jednotlivých metód a modulov ako malých celkov. Výsledkom takéhoto postupu je narastajúci počet týchto testov, ktoré musia byť splnené. Tým je zabezpečená prevencia pred zmenami pôvodnej funkcionality pri pridávaní nových funkcií. Pre testovanie generátora, resp. jeho správanie sa v závislosti na vstupoch slúžia aplikačné testy. V týchto testoch sa overujú oba režimy generátora, teda režim editora aj režim generátora.

5.2 Testovanie konfigurácie

Testovanie vygenerovanej konfigurácie prebiehalo v dvoch fázach. Prvá fáza spočívala v testovaní konfigurácie v tzv. suchom behu programu Ansible, pri ktorom neboli na cieľových zariadeniach vykonávané žiadne zmeny. V tomto režime bolo overené, že konfigurácia je plne funkčná a možno ju využiť pre reálne nasadenie. V druhej fáze to bolo reálne testovanie na simulovaných cieľových zariadeniach, na ktorých bola použitá konfigurácia a manuálne bolo overené nasadenie aplikácií definovaných vo vygenerovanej konfigurácii. Pre účely týchto testov bol vytvorený testovací skript, ktorý umožňoval jednoduché prepínanie režimov programu Ansible bez nutnosti pamätania si prepínačov programu.

Testovací skript

Súčasťou praktického výstupu práce je overovací skript, ktorý na základe prítomnosti parametrov dokáže overiť a spustiť výslednu konfiguráciu. Skript spúšťa hlavný súbor scenára pod názvom `ans_gen.yml` so súborom pre definíciu cieľových zariadení

s názvom `hosts`. Voliteľné parametre skriptu sú:

- *--dry* – definujúce tzv. `suchý` beh programu Ansible, v ktorom overí činnosť programu Ansible z hľadiska prítomnnosti potrebných modulov a najmä správneho zápisu konfigurácie.
- *--debug* – režim Ansible, ktorý obsahuje okrem klasického výpisu aj podrobné hlásenia o stave a vykonávaných činnostiach, čím umožňuje rýchlu identifikáciu problému, vhodné použiť s parametrom `--dry` v prípade, že samotný beh pri prvom pokuse zlyhá.

V prípade spustenia tohto skriptu bez týchto voliteľných parametrov nastane spustenie programu Ansible v normálnom režime a Ansible začne vykonávať jednotlivé inštrukcie definované v konfigurácií.

5.2.1 Návrh testovacieho prostredia

Schéma testovacieho prostredia závisí na zvolenej testovacej konfigurácii. Vzorová konfigurácia s názvom `Simple Lamp Web Server` vyžaduje databázový a webový server (alternatívne jeden hybridný server). S ohľadom na závislosti bolo navrhnuté testovacie prostredie pozostávajúce zo 4 simulovaných staníc s nasledujúcimi funkciami:

- webový server,
- databázový server,
- ansible server,
- testovací klient.

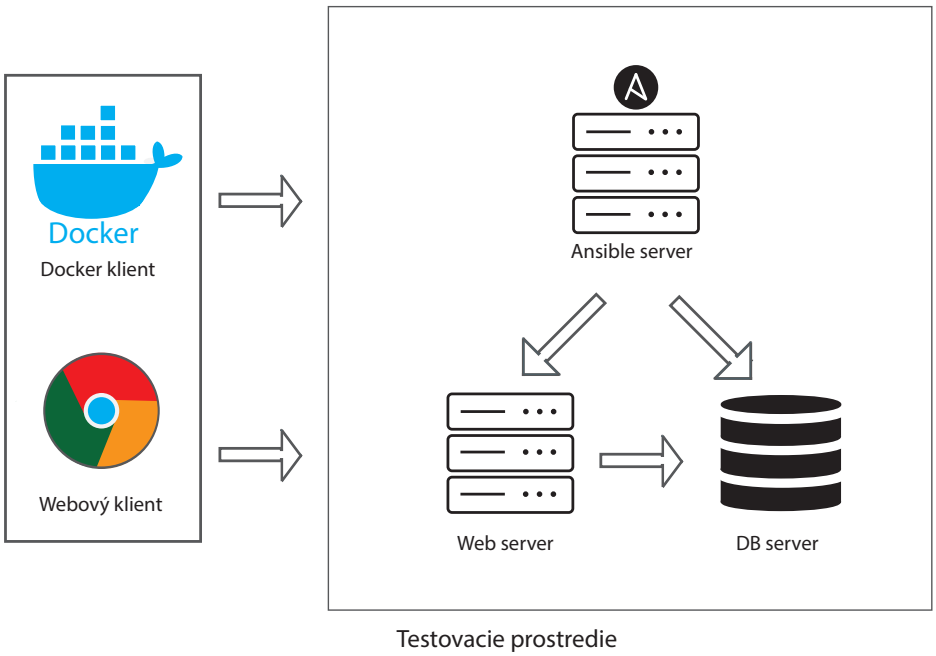
Schému prostredia zobrazuje obr. 5.1.

5.2.2 Realizácia prostredia

Pre realizáciu prostredia je možné využiť virtualizáciu jednotlivých zariadení, alebo ich kontajnerizáciu. Vzhľadom na fakt, že každá stanica má jedinú úlohu, je veľmi výhodné využiť kontajnerizáciu namiesto virtualizácie [35]. Výhodou je najmä rýchlejší štart kontajnera v porovnaní s virtuálnym zariadením a taktiež menšia pamätová a výpočtová náročnosť. Z dôvodu jednoduchého nasadenia, podpory vyžadovaných funkcií a širokej podpory od komunity v podobe balíčkov pre rôzne linuxové operačné systémy bol vybraný nástroj na správu kontajnerov Docker a nástroj Docker Compose, pre rýchle nasadenie skupiny kontajnerov a ich centrálnu správu.

Docker

Pôvodne nástroj na vytváranie a správu kontajnerov, postupne sa stal takpovediac priemyselným štandardnom v oblasti kontajnerizácie a izolovania aplikácií, služieb



Obr. 5.1: Schéma testovacej infraštruktúry.

a pod. Kontajner predstavuje izolované prostredie pre beh jedinej aplikácie či služby bez rizika ovplyvnenia inými aplikáciami, či službami. Tento prístup je veľmi výhodný pre vývojárov, ktorí vyvíjajú v ideálnom prostredí (t.j. v prostredí, ktoré obsahuje len minimálne závislosti), či pre správcov systémov, ktorí nemusia riešiť konfliktné závislosti medzi programami a pod. V porovnaní s virtuálnym strojom má viaceré výhody, najmä však nižšiu pamäťovú a výpočtovú náročnosť, ktoré pramenia z prístupu kontajnera k systémovým zdrojom.

Docker Compose

Docker Compose je nástroj na definovanie a spúšťanie aplikácií zložených z viacerých Docker kontajnerov [36]. Tento prístup využíva textový konfiguračný súbor `docker-compose.yml` s už popísaným formátom YAML, ktorý definuje jednotlivé kontajnery, ich základné atribúty, ako napr.:

- zdrojový priečinok, z ktorého je vybudovaný kontajner, príp. jeho názov, ak je použitý vopred vybudovaný kontajner,
- port na transportnej vrstve, na ktorej je daný kontajner dostupný (voliteľné, vhodné napr. pre web alebo databázový server),
- ďalšie nepovinné parametre, ako napríklad premenné prostredia, pripojiteľné diskové jednotky (angl. `volumes`), či závislosti na ďalších kontajneroch.

V prípade testovania generátora týmto nástrojom je vytvorená testovacia infraštruktúra s využitím nasledujúceho skriptu `docker-compose.yml`:

```
version: '3'
services:
        web:
                build: ./fedora_server/
                image: fedora_server:0.2
                ports:
                        - "1234:22"
        db:
                image: fedora_server:0.2
                ports:
                        - "1235:22"
        ansible:
                build: ./fedora_client/
                image: fedora_client:0.1
                stdin_open: true
                tty: true
```

Takto vytvorená testovacia infraštruktúra obsahuje dvojicu serverov (web a db), bežiacich na operačnom systéme Fedora so zapnutým ssh serverom pre pripojenie Ansible servera a samotný Ansible server. Ten má trvalo otvorený interaktívny terminál pre štandardný vstup, ktorým je možné spúšťať príkazy programu Ansible. Pripojenie ku interaktívnej konzole Ansible kontajnera je možné pomocou príkazu `docker attach <id_kontajnera>`.

Testovací proces

Všetky tri servery (pre web, databázu a ansible) boli kontajnerizované a spravované prostredníctvom nástroja Docker Compose, ktorý sa staral o spúšťanie, správu a korektné vypnutie kontajnerov. Testovací scenár pre manuálne testovanie mal nasledujúce fázy:

1. Štart serverov prostredníctvom nástroja Docker Compose.
2. Prihlásenie sa na Ansible server.
3. Spustenie automatizovanej konfigurácie webového a databázového servera.
4. Kontrola funkčnosti spracovanej konfigurácie pomocou webového prehliadača prostredníctvom testovacieho klienta.

Výsledky manuálneho testovania, spolu s ukážkami automatických testov a vzorových výstupov generátora sú popísané v nasledujúcej kapitole č. 6.

5.3 Statická bezpečnostná analýza

Súčasťou testovania bol bezpečnostný audit kódu pomocou špecializované nástroja. Takýto nástroj nie je schopný odhaliť všetky problémy vyvíjaného nástroja, slúži však ako dobrá spätná väzba pre vývojára. Na základe zoznamu voľne dostupných nástrojov pre statickú analýzu podľa organizácie OWASP[37] boli dostupné 4 nástroje (Bandit[38], LGTM[39], Pyre[40], SonarQube[41]) podporujúce aj programovací jazyk Python. Na základe jednoduchosti nasadenia, používania a podporovaných funkcií vybraný prvý menovaný nástroj `Bandit`. Jedná sa o nástroj dostupný ako zdrojový kód alebo hotové balíčky (napr. v repozitári python knižníc `pypi`) a umožňuje statickú bezpečnostnú analýzu kódu [38]. Nástroj obsahuje desiatky preddefinovaných testov, pokrývajúcich najbežnejšie aktuálne zraniteľnosti, ako napr. používanie zastaraných kryptografických algoritmov a protokolov (md5, telnetlib), alebo používanie nezabezpečených funkcií (input, yaml_load). Umožňuje výber jednotlivých testov pre statickú analýzu, možnosť vynechávania jednotlivých testov alebo tvorbu vlastných testov.

5.3.1 Bezpečnostná analýza implementovaného generátora

Pre účely testovania implementovaného generátora budú použité všetky preddefinované testy, pre spustenie stačí zadať cestu ku kódu a povoliť rekurzívne prehľadávanie priečinkov nasledujúcim príkazom:

```
bandit --recursive ~/program
```

Prevedenou statickou analýzou kódu bolo zistených 5 rôznych bezpečnostných zraniteľnosti. Prvým a jedným z najzávažnejších nedostatkov bolo používanie funkcie `eval()` na vyhodnocovanie funkcií uložených v textovom reťazci, ako zobrazuje nasledujúci výpis:

```
>> Issue: [B307:blacklist] Use of possibly insecure function - consider using
↪  safer ast.literal_eval.
        Severity: Medium   Confidence: High
        Location: /home/mmatisko/program/rules/iterator_regex.py:23
        22          regex_str = regex_str.replace('#', str(self.__iteration))
        23          return int(eval(regex_str))
```

Táto funkcia však neoveruje textový reťazec pred jeho vykonaním, takže potenciálne umožňuje spustenie škodlivého kódu pri vyhodnocovaní[1]. Výsledkom škodlivého kódu môže byť pád programu bez akéhokoľvek upozornenia používateľa, resp.

[1]Zraniteľnosť funkcie eval() je podrobnejšie popísaná na <`https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html`>.

zaznamenania situácie, či dokonca deštrukčná operácia. Príklad zneužitia takejto zraniteľnosti, zobrazuje nasledúci príkaz:

```
eval("__import__('os').system('rm -rf /')")
```

Dôsledky tohoto jednoduchého vstupu sú kritické, použitím funkcie eval dôjde k vymazaniu dát a znefunkčneniu zariadenia. Túto zraniteľnosť však možno jednoducho vyriešiť použitím funkcie `compile()` a jej výstup použiť ako parameter funkcie `eval()`, alebo použitím funkcie `ast.literal_eval()`[42]. Táto funkcia v prípade vyššie uvedeného škodlivého vstupu vytvorí chybovú hlášku, ktorou upozorní používateľa, nehrozí teda vymazanie dát.

Menej závažnou zraniteľnosťou v tomto konkrétnom prípade je nasledujúca zraniteľnosť spočívajúca v používaní pseudo-náhodného generátora pre bezpečnostné resp. kryptografické účely.

```
>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for
↪  security/cryptographic purposes.
        Severity: Low   Confidence: High
        Location: /home/mmatisko/program/rules/list_reader.py:116
        115         while random_item == -1 or random_item in
↪           self.__used_items:
        116             random_item = randint(0, item_count - 1)
        117             self.__used_items.add(random_item)
```

Správnym prístupom je nahradenie pôvodnej funkcie `random.randint()` generátorom náhodných čísel určený pre použitie v kryptografii `os.urandom()`[43]. Táto funkcia vracia sekvenciu náhodných bytov v závislosti na zdroji náhodnosti operačného systému (napr. `/dev/urandom` v prípade linuxových systémov).

Nasledujúce dve zraniteľnosti patria do spoločnej kategórie tzv. výskytov hesiel v kóde. Prvou z nich je zraniteľnosť s označením B105, ktorá označuje heslo zapísané v kóde programu ako reťazec znakov, čo ilustruje nasledujúci výpis:

```
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password:
↪  'password'
        Severity: Low   Confidence: Medium
        Location: /home/mmatisko/program/test/unit_test.py:326
        325
        326         password = 'password'
        327         vault = FileVault(filepath=dst_path, password=password)
```

Druhá zraniteľnosť s označením B106 označuje využitie tohto reťazca, ktorý obsahuje heslo, ako parameter funkcie, ako možno vidieť v nasledujúcom výpise:

```
>> Issue: [B106:hardcoded_password_funcarg] Possible hardcoded password:
↪  'password'
        Location: /home/mmatisko/program/test/app_test.py:85
        84
        85 password = 'password'
        86 test_conf = Configuration(path=os.path.join(working_dir, 'group_vars',
↪  'all'), password=password)
```

Kedže je výskyt dvoch uvedených zraniteľností, popisujúcich výskyt hesla v kóde, výlučne v súboroch obsahujúcich výlučne testy a jedná sa len o testovacie (teda neprodukčné) heslá a konfigurácie, je možné tieto dve zraniteľnosti akceptovať.

Po oprave vyššie uvedených zraniteľností možno opäť spustiť nástroj `Bandit`, tentokrát bez testov na zraniteľnosti, ktoré súvisia len s testovacími súbormi:

```
bandit --recursive ~/program/ --skip B105,B106
```

Následný výstup statickej analýzy overí správne aplikované opravy detegovaných zraniteľností:

```
[main]        INFO        profile include tests: None
[main]        INFO        profile exclude tests: None
[main]        INFO        cli include tests: None
[main]        INFO        cli exclude tests: B105,B106
[main]        INFO        running on Python 3.7.7
Run started:2020-05-24 19:45:23.992075

Test results:
No issues identified.

Code scanned:
Total lines of code: 1409
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
Undefined: 0.0
Low: 0.0
Medium: 0.0
High: 0.0
Total issues (by confidence):
Undefined: 0.0
Low: 0.0
Medium: 0.0
High: 0.0
Files skipped (0):
```

6 Výsledky testovania

Súčasťou praktickej časti tejto práce bolo testovanie implementácie. Táto kapitola rozoberá výsledky testovania generovanej konfigurácie, teda praktické nasadenie na virtuálnu testovaciu infraštruktúru. V druhej časti sú to výsledky časovej náročnosti pre rastúce množstvo generovaných konfigurácií a taktiež praktické overenie optimalizácie generovania využitím paralelizácie použitím modulu `Threadpool`.

6.1 Dosiahnuté výstupy

Nasledujúca podkapitola zobrazuje jednotlivé výstupy pre oba režimy implementovaného nástroja, editačný režim aj režim generátora. Výsledky zobrazujú všetky implementované možnosti daného režimu pre platné vstupy (s výnimkou overenia neplatných vstupov).

6.1.1 Editačný režim

Vyvinutý generátor podporuje spomínané vlastnosti editačného režimu, vrátane zadávania statických hodnôt a generovania hodnôt zo zadanej siete, rozsahu portov či údajov uložených v textovom súbore.

Príklad generovania IP adresy zo zadanej siete, pôvodná konfigurácia obsahovala zoznam premenných:

```
server_ip: 192.168.21.12
server_port: 3048
server_login: admin
server_pass: admin
```

Po spustení generátora v editačnom režime so zadanou sieťou `192.168.22.0/24` bola výsledkom konfigurácia so zmenenou premennou `server_ip`:

```
generator.py -e -n 192.168.22.0/24 -i server_ip ./config/
generator.py -e -p 1000-5000 -i server_port ./config/
```

Generátor otvorí zložku `config`, prehľadá konfiguráciu a v súbore `group_vars/webservers` upraví hodnotu server_ip na hodnotu z požadovaného rozsahu, výsledný súbor bude vyzerať nasledovne:

```
server_ip: 192.168.22.127
server_port: 3333
server_login: admin
server_pass: admin
```

Zmena prihlasovacích parametrov pomocou výberu z textového súboru je možná nasledovným spôsobom:

```
generator.py -e -f ./logins.csv -i server_login ./config/
generator.py -e -f ./passwords.csv -i server_pass ./config/
```

Výsledkom budú náhodne vybrané parametre `server_login` a `server_pass`, ktoré nahradia doterajšie preddefinované hodnoty:

```
server_ip: 192.168.22.127
server_port: 3333
server_login: user3675
server_pass: P4&34Hgf3
```

6.1.2 Režim generátora

V režime generátora je okrem CLI argumentov využívaný aj konfiguračný súbor, výsledné parametre generátora preto závisia na kombinácii oboch týchto zdrojov parametrov, pričom vyššiu prioritu majú CLI argumenty. Kombinácia nasledujúceho konfiguračného skriptu:

```
# General settings
general:
- iterations: 10
  name: YAML generator
  input: /home/user/nginx_ha_debian/
  output: /home/user/output/

# Static values
static:
- machine_name: "AlabamaCluster<#+100>"
  admin_accout: "ClusterAdmin"
  network_mask: 255.255.255.0
  default_gateway: 192.168.1.254

# Dynamic values
dynamic:
- ip_address: 192.168.<#>.0/25
  admin_pass: /home/user/downloads/rockyou.txt
```

a použitím nižšie uvedených CLI argumentov:

```
generator.py -g -c ../generator_config.yml -d /home/administrator/nginx_ubuntu/
↪  -o /home/administrator/configs/
```

dáva k dispozícií dva priečinky so vstupnou šablónovou konfiguráciou, podobne aj dva výstupné priečinky pre vygenerované konfigurácie. Pretože vyššiu prioritu

majú CLI argumenty, ako už bolo uvedené, bude vnútorná reprezentácia nastavení generátora vyzerať nasledovne:

```
{
        "mode": "Generate",
        "input": "/home/administrator/nginx_ubuntu/",
        "output": "/home/administrator/configs/",
        "general":
        {
                "iterations": "10",
                "name": "Nginx web cluster"
        },
        "static":
        {
                "machine_name": "AlabamaCluster<#+100>",
                "admin_account": "ClusterAdmin",
                "default_gateway": "192.168.<#>.254"
        },
        "dynamic":
        {
                "ip_address": "192.168.<#>.0/25",
                "admin_pass": "/home/user/downloads/rockyou.txt"
        }
}
```

Takto nastavený generátor vygeneruje 10 výstupných konfigurácií a uloží ich do priečinka označeného časovou pečiatkou (angl. `timestamp`). Úplná cesta vygenerovaných konfigurácií má tvar `/home/administrator/configs/18-05-2020_16:06:46/`. Výslednú konfiguráciu vygenerovanú v prvej iterácii demonštruje nasledujúci výpis:

```
# Server settings
machine_name: AlabamaCluster100
admin_account: ClusterAdmin
admin_pass: ROCK_YOU_PASSWORD

# Network settings
ip_address: 192.168.0.123/25
network_mask: 255.255.255.0
default_gateway: 192.168.1.254
ssh_port: 22
ntp_port: 123
```

Vygenerované hodnoty, zvýraznené zelenou farbou, sú získané zo statických a dynamických pravidiel. Všeobecný kľúč `name` je využitý len pre informatívne účely a logovanie.

6.2 Testovanie optimalizácie

V nasledujúcej časti sú zoradené výsledky testovaní časovej a pamäťovej náročnosti nešifrovaných vstupných súborov a nešifrovaných generovaných konfigurácií. Následne bola meraná časová náročnosť zvyšovaním počtu vykonávaných vlákien. Teoretický predpoklad hovorí, že zrýchlenie by malo narastať do počtu podporovaných vlákien fyzického procesora a následne by malo potenciálne zrýchlenie klesať. V druhej fáze tohto testovania bol meraný a porovnávaný vplyv šifrovania pomocou programu Ansible Vault pre nasledujúce tri scenáre:

- porovnanie nešifrovaného a šifrovaného konfiguračného súboru,
- porovnanie nešifrovanej a šifrovanej výstupnej konfigurácie (pri použití nešifrovanej šablónovej konfigurácie),
- porovnanie nešifrovanej a šifrovanej šablónovej a vygenerovanej konfigurácie.

6.2.1 Testovací postup

Výsledky testovacích scenárov, uvedené v nasledujúcich podkapitolách, sú priemerné dosiahnuté vykonávacie časy pre danú kombináciu vstupných parametrov. Pre zníženie vplyvu externých rušení, napr. plánovača operačného systému, prebiehal každý test 5 ×, pričom boli odstránené extrémne hodnoty a bol vypočítaný priemerný výsledok troch zostávajúcich hodnôt. V prípade, ak boli veľké rozdiely medzi jednotlivými vykonávacími časmi (rozptyl vyšší ako 5 % od prostrednej hodnoty), bol test opakovaný. Spúšťanie potrebného počtu opakovaní testu, vrátane potrebného hesla pre ansible konfiguráciu zabezpečil nasledujúci parameter:

```
for i in {1..5}; do time ./testing_runner.sh 'password' ./production_ans_gen.sh
↪  -g -c include/testing_generator_config_enc.yml; done
```

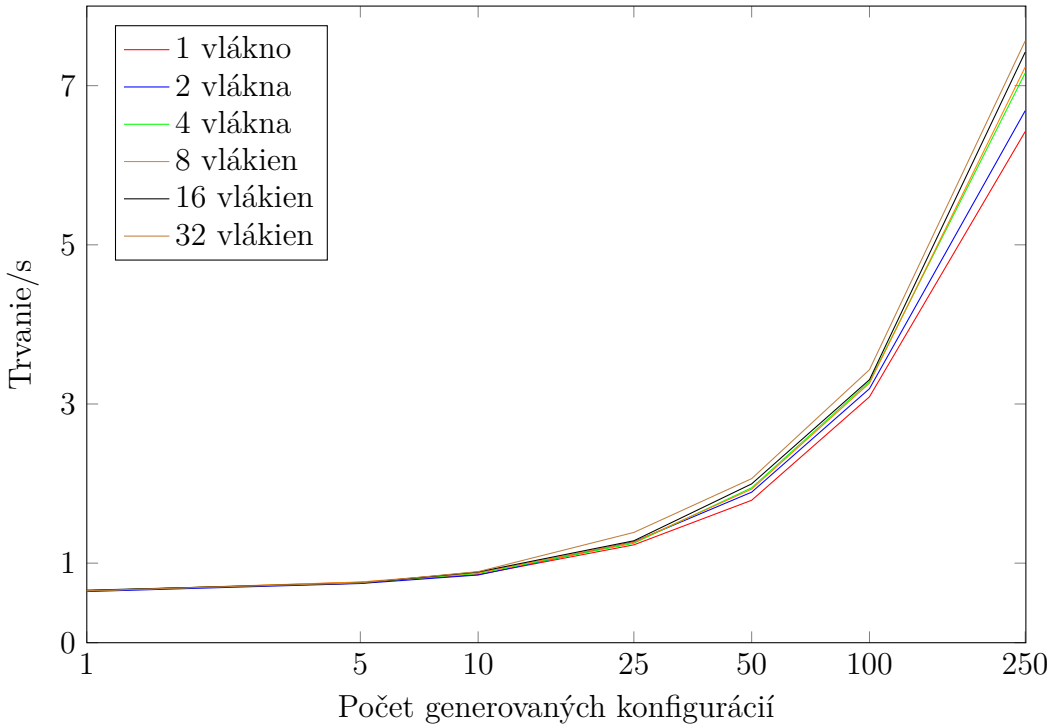
slučka zabezpečuje potrebný počet opakovaní, zabudovaná funkcia terminálového klienta `time` zabezpečuje meranie vykonávacieho času. Tá spúšťa pripravený skript, dostupný v prílohe B, ktorý je postavený na nástroji expect a umožňuje automatizovane odpovedať na výzvy programu vložením hesla bez zdržania spôsobeného manuálnym zadávaním testovacieho hesla (keďže použitá funkcia na načítavanie hesla `getpass` využíva interaktívnu formu terminálu `/dev/tty/` namiesto štandardného konzolového vstupu `stdin`). Výnimkou pre používanie tohto skriptu boli testy trvajúce <1s, pri ktorých nastávala kolízia vytváraného priečinka pre výstupnú konfiguráciu.

6.2.2 Scenár testovania časovej náročnosti

Testovanie časovej náročnosti prebiehalo v terminálovom klientovi `bash`. Pre účely tohto testovania je použitý prázdny vstup, keďže sa jedná o testovanie nešifrovanej

konfigurácie (vstupnej šablónovej aj výstupnej generovanej). Následne bolo zvolených 7 hodnôt, ktoré udávajú počet výstupných konfigurácií, od minimálneho počtu 1 konfigurácie až po takmer maximálnu hodnotu. Maximálny možný počet parametrov závisí vždy na kombinácií zadaných vstupov, v prípade tohto testovania je to počet sietí, teda maximálne 256 výstupných konfigurácií.

Celé testovanie pre každý počet výstupných konfigurácií prebiehalo sekvenčne a výsledná hodnota je aritmetickým priemerom 3 po sebe idúcich meraní. Výsledný graf tohto testovania je na obr. 6.1, presné hodnoty uvádza v prílohe tabuľka C.1. Prvým krokom bolo testovanie na jednom generujúcom vlákne pre všetky zvolené počty výstupných konfigurácií (zvolené hodnoty sú 1, 5, 10, 25, 50, 100 a 250). Tieto časy sú považované za referenčné. Následne bol počet generujúcich vlákien zvyšovaný až do počtu, pri ktorom nastal nárast času potrebného na generovanie konfigurácií, v porovnaní s dvojicou predošlých hodnôt. Celkovo toto testovanie prebehlo pre 1, 2, 4, 8, 16 a 32 vlákien.



Obr. 6.1: Graf trvania generovania konfigurácií pre rôzny počet vlákien.

Z výsledkov tohto testovania je možné vidieť, že generovanie konfigurácie v nešifrovanej forme je veľmi rýchle a zvyšovanie počtu vlákien nespôsobuje zrýchlenie, ale naopak mierne spomalenie vykonávacieho času. Rozdiel medzi najrýchlejším priemerným nameraným časom (6,423 s pri 1 vlákne) a najpomalším časom (7,562 s pre 32 vlákien) je 1,239 s, čo predstavuje podiel 19,3 % najrýchlejšieho priemerného

vykonávacieho času pre maximálny testovací počet 250 výstupných konfigurácií. Dôvodom spomalenia je kombinácia viacerých faktorov, najvýraznejší vplyv majú réžia nad vláknami a synchronizácia prideľovania úloh medzi jednotlivé úlohy, čo zabezpečuje plánovač operačného systému a taktiež nezanedbateľný vplyv má aj použitie mechanického disku, ktorý vytvára určite spomalenie prameniace z odozvy disku pri paralelnom zápise.

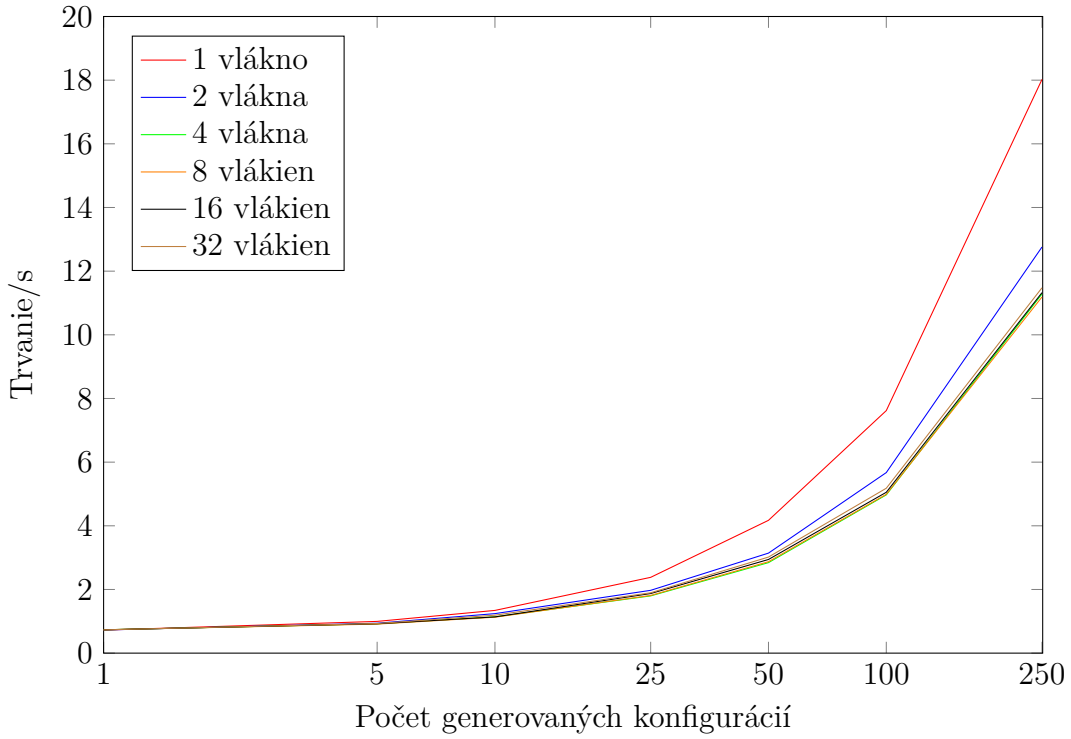
6.2.3 Scenáre testovania vplyvu šifrovania

Nasledujúce dva scenáre boli takmer totožné s prvým scenárom (boli použité rovnaké hodnoty počtu vygenerovaných konfigurácií, bola použitá rovnaká šablónová konfigurácia), zmenou oproti predošlému scenáru bolo použitie šifrovania využitím nástroja Ansible Vault. V prvom z týchto scenárov bola šifrovaná citlivá časť vygenerovaných konfigurácií použitím nešifrovanej vstupnej šablónovej konfigurácie. V druhom scenári bola využitá už šifrovaná šablónová konfigurácia a rovnako šifrované boli aj vygenerované konfigurácie.

Nešifrované vstupné súbory vs šifrovaný konfiguračný súbor

Prvý scenár s využitím nástroja Ansible Vault bol scenár so šifrovaným výstupom, konfiguračný súbor pre generátor a vstupná šablónová konfigurácia je nešifrovaná. Výsledok tohto testovania zobrazuje obr. 6.2, presné hodnoty v prílohe uvádza tabuľka D.1.

Ako je možné vidieť na výslednom grafe, najvýraznejšie zrýchlenie bolo medzi samostatným vláknom na generovanie a medzi dvomi generujúcimi vláknami. Najvýraznejšie toto zrýchlenie vidieť na maximálnom množstve generovaných konfigurácií, kde vykonávací čas pre samostatné vlákno dosiahol priemernú hodnotu 18,024 s a vykonávací čas pre dvojicu vlákien dosiahol priemernú hodnotu 12,759 s. To predstavuje zrýchlenie na úroveň 70,79 % pôvodného vykonávacieho času. Najrýchlejším priemerným vykonávacím časom bola hodnota pre 8 vlákien, ktorá dosiahla hodnotu 11,190 s, čo predstavuje zrýchlenie na úroveň 62,08 % pôvodného vykonávacieho času. Dôvodom, prečo sa časy neblížia maximálnemu teoretickému zrýchleniu, ktoré je dané počtom využitých vlákien, je najmä nízka komplexnosť generovanej konfigurácie. Keďže však dochádza k šifrovaniu konfigurácie počas zápisu na disk (v tomto prípade sa jedná o mechanický disk, ktorý pridáva určité spomalenie pri zápise), k zrýchleniu v tomto scenári na rozdiel od scenára pre nešifrovaný výstup už dochádza.

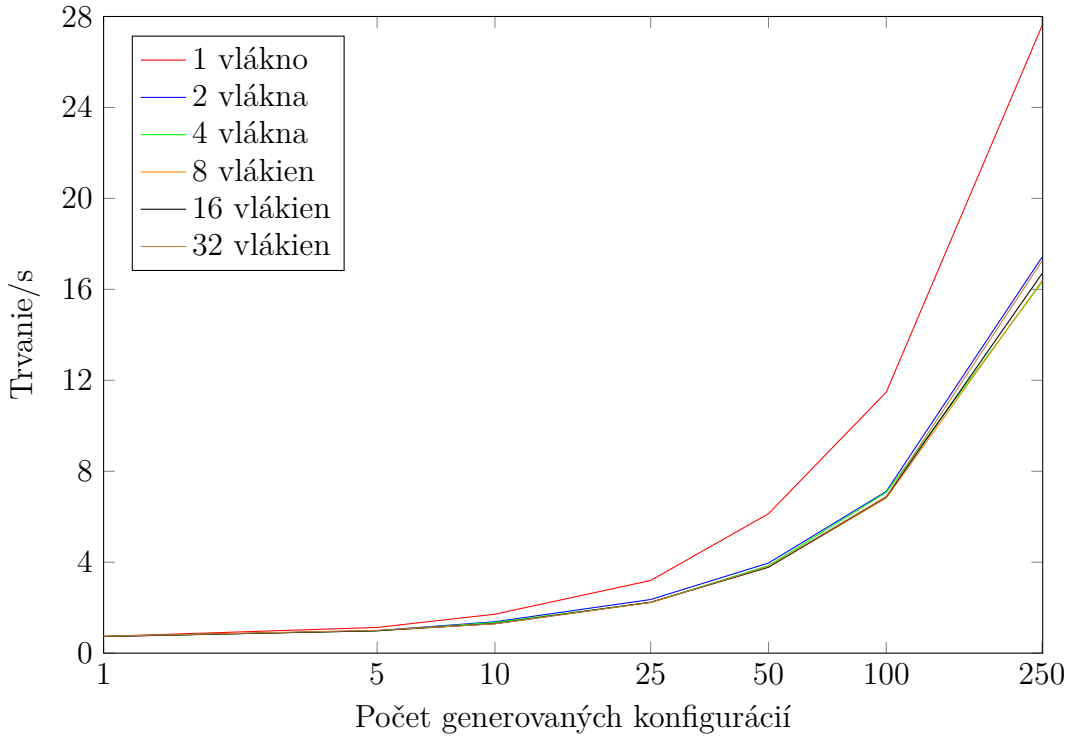


Obr. 6.2: Graf generovania šifrovaných konfigurácií použitím šifrovanej šablóny pre rôzny počet vlákien.

Nešifrované vstupné súbory vs šifrovaná výstupná konfigurácia

Druhý scenár využívajúci Ansible Vault pracuje so šifrovanou vstupnou konfiguráciou a generuje šifrované konfigurácie. Keďže režim generátora pracuje nad kópiou šifrovanej šablónovej konfigurácie, tento scenár, v porovnaní s predchádzajúcim obsahuje aj dešifrovanie pri čítaní. Vplyv dešifrovania a šifrovania na vykonávací čas zobrazuje obr. 6.3. Konkrétne hodnoty sú uvedené v prilohe v tabuľke D.2.

V prípade šifrovanej šablónovej konfigurácie možno z grafu vyčítať menší rozdiel v priemerných vykonávacích časoch medzi dvomi a viacerými vláknami. Pre maximálne množstvo generovaných konfigurácií dosahuje priemerný vykonávací čas na jednom vlákne 27,590 s, generovanie na 32 vláknach dosiahlo priemerný vykonávací čas 17,245 s. Najlepší vykonávací čas 16,309 s dosiahlo v testovaní 8 vlákien, čo predstavuje zrýchlenie na úroveň 59,11 % pôvodného vykonávacieho času. Toto priemerné zrýchlenie je takmer 3 % vyššie ako v predchádzajúcom scenári, čo potvrdzuje vyššiu náročnosť na výkon procesora v tomto scenári. Tento fakt potvrdzujú aj vyrovnané výsledky medzi vyšším počtom vlákien, kedy režijné náklady na prevádzku vlákien kompenzujú samotné zrýchlenie, avšak nedochádza v tomto prípade k výraznejšiemu spomaleniu, najmä v pomere k samostatnému vláknu.



Obr. 6.3: Graf šifrovaných konfigurácií použitím nešifrovanej šablóny pre rôzny počet vlákien.

6.3 Porovnanie testovacích scenárov

V podkapitole 6.2 boli prezentované výsledky optimalizácie generovania konfigurácií a vplyv šifrovania a dešifrovania na vykonávací čas. V tejto kapitole sú vybrané a porovnané časy naprieč týmito scenármi, kde je možné vidieť vplyv šifrovania a šifrovania + dešifrovania na vykonávací čas. Nasledujúce tabuľky obsahujú vybrané hodnoty z uvedených testovacích scenárov, uvedených v prílohe D, v stĺpcovom prehľade. Prvý stĺpec udáva počet iterácií, teda generovaných konfigurácií, nasledujúce tri stĺpce uvádzajú priemerné vykonávacie časy v danom scenári pre určený počet vlákien a posledné dva stĺpce udávajú percentuálny pomer časov v scenári č. 2, resp. scenára č. 3, v porovnaní so scenárom č. 1. V oboch tabuľkách vidíme postupný nárast pomeru vykonávacích časov. To potvrdzuje vyššie uvádzané tvrdenie, že s rastúcim počtom generovaných konfigurácií a rastúcou náročnosťou na generovanie alebo čítanie / zápis (do) súboru, rastie efektivita využitia viacerých vlákien. Porovnanie pre jedno využité vlákno zobrazuje tabuľka č. 6.1.

Možno vidieť, že časová náročnosť pri generovaní 250 šifrovaných výstupných konfigurácii vzrástla takmer trojnásobne, zatiaľ čo časová náročnosť generovania šifrovaných konfigurácií zo šifrovanej šablónovej konfigurácie vzrástla viac ako štvornásobne.

Tab. 6.1: Porovnanie scenárov využitím 1 vlákna.

počet iterácií	Scenár 1	Scenár 2	Scenár 3	Nárast 2 vs 1	Nárast 3 vs 1
1	0,651 s	0,723 s	0,736 s	110,59 %	113,06 %
10	0,856 s	1,342 s	1,381 s	156,78 %	161,33 %
25	1,230 s	2,381 s	3,195 s	192,58 %	259,76 %
100	3,090 s	7,619 s	11,479 s	246,57 %	371,49 %
250	6,423 s	18,024 s	27,590 s	280,62 %	429,55 %

Nasledujúca tabuľka č. 6.2 porovnáva priemerne najrýchlejšie vykonávané časy, ktoré boli namerané na 8 využitých vláknach. Možno vidieť, že časová náročnosť generovania 250 šifrovaných konfigurácií výrazne poklesla na približne 1,5 násobok vykonávacieho času nešifrovaného výstupu. Pri využití šifrovanej šablónovej konfigurácie bol vykonávací čas viac ako dvojnásobný, to je však takmer polovičná hodnota oproti porovnaniu pri využití jedného vlákna.

Tab. 6.2: Porovnanie scenárov využitím 8 vlákien.

počet iterácií	Scenár 1	Scenár 2	Scenár 3	Nárast 2 vs 1	Nárast 3 vs 1
1	0,652 s	0,734 s	0,739 s	112,58 %	113,34 %
10	0,879 s	1,129 s	1,297 s	128,44 %	147,55 %
25	1,264 s	1,812 s	2,227 s	143,35 %	176,19 %
100	3,261 s	4,989 s	6,821 s	152,99 %	209,17 %
250	7,234 s	11,190 s	16,375 s	154,69 %	226,36 %

Záver

Zadaním diplomovej práce bolo vytvoriť generátor konfigurácie pre nástroj automatickej konfigurácie (pre zvolený nástroj Ansible) podľa definovaných kritérií. Kritériami sú rôzne rozsahy sietí, statické IP adresy, porty či prihlasovacie údaje. Aplikácia mala byť vyvíjaná v jazyku Python a podporovať rozšírenia o ďalšie konfiguračné nástroje.

V prvej kapitole teoretickej časti práce bola rozobraná problematika sieťovej automatizácie, hlavné výhody a nevýhody použitia automatizácie a ponúkané možnosti. Boli popísané kritéria výberu nástroja, uvedené existujúce riešenia a podporné protokoly využívané v sieťovej automatizácií. Druhá kapitola pojednáva o prístupe nazývanom Infraštruktúra ako kód, ktorá je základom nástrojov sieťovej automatizácie. Boli uvedené prístupy správy konfigurácie a ich porovnanie z hľadiska použitia. Vzhľadom na zameranie diplomovej práce na nástroj Ansible, bol tento nástroj popísaný so zameraním na jeho fungovanie, možnosti ktoré ponúka a primárne na jeho konfiguračné súbory, ktoré sú výstupom tejto práce.

Praktická časť diplomovej práce začína v kapitole 3, ktorá sa venuje návrhu generátora konfigurácie. Bola definovaná základná požadovaná funkcionalita generátora. Na jej základe boli definované režimy generátora a prostredníctvom nich bola navrhnutá vnútorná štruktúra generátora, s ohľadom na budúcu rozšíriteľnosť a prehľadnosť programového kódu. Štvrtá kapitola popisuje implementáciu generátora a výslednú štruktúru, ktorá bola upravená v priebehu implementácie oproti teoretickému návrhu v tretej kapitole.

Predposledná piata kapitola sa venuje testovaniu generátora, popisuje typy testov ktoré boli využité v priebehu implementácie a po jej dokončení. Obsahuje návrh virtuálneho prostredia, ktoré slúži na manuálne testovanie vygenerovanej konfigurácie a použité technológie pre realizáciu tohto prostredia. Posledná šiesta kapitola agreguje vzorové výstupy oboch režimov generátora (editačného režim aj režimu generátora) a výsledky automatického testovania pre tri rôzne scenáre. Prvý scenár bol zameraný na možné optimalizácie využitím paralelizácie (použitím väčšieho počtu vlákien) pri generovaní. Druhý a tretí scenár testoval vplyv šifrovania resp. dešifrovania na celkový čas behu generátora. Posledná časť kapitoly obsahuje porovnanie výsledkov jednotlivých scenárov a optimálne nastavenie na testovacom zariadení.

Výsledkom diplomovej práce je generátor podporujúci dvojicu funkčných režimov. V editačnom režime dokáže na základe definovaných kritérií upravovať už existujú konfiguráciu a je možné ho prevádzkovať bez hlbšej znalosti vnútorného usporiadania a fungovania Ansible konfigurácie. V tomto režime nástroj funguje s dvojicou zadaných parametrov, názvom kľúča resp. premennej a novou hodnotou pre daný kľúč. Novou hodnotou možno rozumieť staticky zadaný vstup alebo roz-

sah hodnôt, ktorý je možné zadať manuálne ako parameter generátora alebo ako zdrojový textový súbor s hodnotami. Generátor následne vyberie náhodnú hodnotu z rozsahu alebo zoznamu hodnôt a touto vybranou hodnotou nahradí pôvodnú hodnotu v šablóne.

Pokročilejším režimom nástroja je režim generátor, v ktorom sa využíva konfiguračný súbor generátora obsahujúci jednotlivé pravidlá pre generovanie a šablónová konfigurácia, ktorá slúži ako základ a je následne upravovaná. Na rozdiel od editačného režimu je v tomto režime zachovaná pôvodná konfigurácia bezo zmeny. Oba režimy podporujú šifrovanú konfiguráciu pomocou nástroja Ansible Vault. Tento generátor bol otestovaný a výsledky testovania sú súčasťou tejto práce.

Uvedený nástroj bude použitý na tvorbu testovacích scenárov pre počítačové cvičenia. V spolupráci s automatizačným nástrojom Ansible umožňuje zo zadanej šablónovej konfigurácie vytvoriť desiatky až stovky originálnych scenárov, v závislosti na jednotlivých nastaveniach. Scenáre sa od seba môžu líšiť takmer akýmkoľvek nastavením, od sieťových adries jednotlivých zariadení v neznámej cieľovej sieti, transportnými portami, náhodnými prihlasovacími údajmi, doménovými názvami a pod. Každý študent teda bude v rámci cvičenia pracovať na probléme v unikátnom prostredí.

Literatúra

[1] Juniper Networks *What is network automation?* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://www.juniper.net/us/en/products-services/what-is/network-automation/>.

[2] TechTarget *Network automation* [online]. Network management and monitoring: The evolution of network control [cit. 13. 10. 2019]. Dostupné z URL: <https://searchnetworking.techtarget.com/definition/network-automation>.

[3] Itential *Itential Network Automation Use Cases* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://www.itential.com/solutions/automation-use-cases/>

[4] ELGART, Robert *How to Choose the Right Network Automation Tools* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <http://www.turn-keytechnologies.com/blog/network-solutions/how-to-choose-the-right-network-automation-tools>.

[5] PERKIN, Roger *Free Network Automation Tools* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://www.rogerperkin.co.uk/network-automation-tools/>.

[6] ARORA, Shivam *What Is Ansible?* [online]. [cit. 20. 12. 2019]. Dostupné z URL: <https://www.simplilearn.com/what-is-ansible-article/>.

[7] Puppet *How Puppet works* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://puppet.com/products/how-puppet-works>.

[8] Chef *Chef Automate* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://www.chef.io/products/automate/>.

[9] Saltstack docs *Introduction to Salt* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://docs.saltstack.com/en/latest/topics/>.

[10] Jenkins *Jenkins User Documentation* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://jenkins.io/doc/>.

[11] ManageEngine *SNMP tutorial* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://www.manageengine.com/network-monitoring/what-is-snmp.html>.

[12] Solarwinds *What is Netflow?* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://www.solarwinds.com/netflow-traffic-analyzer/use-cases/what-is-netflow>.

[13] Noviflow *The basics of SDN and the OpenFlow Network Architecture* [online]. [cit. 13. 10. 2019]. Dostupné z URL: <https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/>.

[14] Noviflow *SSH Command* [online]. [cit. 14. 10. 2019]. Dostupné z URL: <https://www.ssh.com/ssh/command/>.

[15] MORRIS, Kief *Infrastructure as Code*[online]. [cit. 23. 10. 2019]. ISBN: 9781491924334. Dostupné z URL: <https://www.oreilly.com/library/view/infrastructure-as-code/9781491924334/ch01.html>.

[16] MORRIS, Kief *Infrastructure as Code*[online]. [cit. 23. 10. 2019]. Dostupné z URL: <https://infrastructure-as-code.com/>.

[17] GUCKENHEIMER, Mike *What is Infrastructure as Code?* [online]. [cit. 23. 10. 2019]. Dostupné z URL: <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code>.

[18] CHAN, Mike *15 Infrastructure as Code tools you can use to automate your deployments* [online]. [cit. 23. 10. 2019]. Dostupné z URL: <https://www.thorntech.com/2018/04/15-infrastructure-as-code-tools/>.

[19] Cloudify *Orchestration for Configuration Management Tools* [online]. [cit. 23. 10. 2019]. Dostupné z URL: <https://cloudify.co/configuration-management-and-orchestration>.

[20] BOERSMA, Eric *Infrastructure as Code: What Is It, and Why Should My Engineers Care?* [online]. [cit. 23. 10. 2019]. Dostupné z URL: <https://www.plutora.com/blog/infrastructure-as-code>.

[21] BRIKMAN, Yevgeniy *Why we use Terraform and not Chef, Puppet, Ansible, SaltStack, or CloudFormation* [online]. [cit. 23. 10. 2019]. Dostupné z URL: <https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c#64a6>.

[22] Ansible *Integration: Infrastructure Automation with Ansible* [online]. [cit. 24. 10. 2019]. Dostupné z URL: <https://www.ansible.com/integrations/infrastructure>.

[23] Datanyze *Market Share: Configuration Management* [online]. [cit. 24. 10. 2019]. Dostupné z URL: <https://www.datanyze.com/market-share/configuration-management>.

[24] Ansible Docs *User guide: Working with modules – Introduction* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/user_guide/modules_intro.html>.

[25] Ansible Docs *User guide: Working with modules – Module Maintenance & Support* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/user_guide/modules_support.html>.

[26] Yaml *YAML* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://yaml.org>.

[27] Ansible Docs *YAML Syntax* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html>.

[28] Tutorials Point *YAML - Basics* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://www.tutorialspoint.com/yaml/yaml_basics.htm>.

[29] Jinja *Template Designer Documentation* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://jinja.palletsprojects.com/en/2.10.x/templates/>.

[30] Ansible docs *User Guide: Working With Playbooks: Introduction* [online]. [cit. 26. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html>.

[31] Ansible docs *User Guide: Working With Playbooks: Including and Importing* [online]. [cit. 26. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_includes.html>.

[32] Ansible docs *User Guide: Working With Playbooks: Roles* [online]. [cit. 27. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html>.

[33] Ansible docs *User Guide: Ansible Vault* [online]. [cit. 27. 10. 2019]. Dostupné z URL: <https://docs.ansible.com/ansible/latest/user_guide/vault.html>.

[34] Agile Aliance *Glossary: TDD* [online]. [cit. 9. 5. 2020]. Dostupné z URL: <https://www.agilealliance.org/glossary/tdd/>.

[35] Docker *What is a Container?* [online]. [cit. 10. 5. 2020]. Dostupné z URL: <https://www.docker.com/resources/what-container>.

[36] Docker docs *Overview of Docker Compose* [online]. [cit. 10. 5. 2020]. Dostupné z URL: <https://docs.docker.com/compose/>.

[37] Owasp *Source Code Analysis Tools* [online]. [cit. 25. 5. 2020]. Dostupné z URL: <https://owasp.org/www-community/Source_Code_Analysis_Tools>.

[38] *Bandit Github repository* [online]. [cit. 25. 5. 2020]. Dostupné z URL: <https://github.com/PyCQA/bandit>.

[39] *About LGTM* [online]. [cit. 26. 5. 2020]. Dostupné z URL: <https://lgtm.com/help/lgtm/about-lgtm>.

[40] *PYre Getting Started* [online]. [cit. 26. 5. 2020]. Dostupné z URL: <https://pyre-check.org/docs/installation.html>.

[41] *SonarQube Docs* [online]. [cit. 26. 5. 2020]. Dostupné z URL: <https://docs.sonarqube.org/latest/>.

[42] Python docs *ast — Abstract Syntax Trees* [online]. [cit. 25. 5. 2020]. Dostupné z URL: <https://docs.python.org/3.7/library/ast.html>.

[43] Python docs *os — Miscellaneous operating system interfaces* [online]. [cit. 25. 5. 2020]. Dostupné z URL: <https://docs.python.org/3.7/library/os.html>.

Zoznam symbolov, veličín a skratiek

AES Advanced Encryption Standard – pokročilý šifrovací štandard

CDN Content Delivery Network – sieť serverov určená pre doručovanie dát veľkému množstvu klientov

CI/CD Continous Integration / Continous Delivery – spôsob vývoja softvéru pomocou automatizácie, typicky sa jedná o automatizované spúšťanie, testovanie a nasadenie

CLI Command line interface – textové používateľské rozhranie

CSV Comma separated values – textový formát, ktorý oddeľuje jednotlivé záznamy pomocou čiarky (,)

HTML HypterText Markup Language – značkovací jazyk, používaný pri zápise web stránok, vhodný na prenos

HTTP HyperText Transport Protocol – protokol používaný na prenos dokumentov, typicky v smere od webservera ku klientovi

IaC Infrastructure as Code – proces správy a nasadenia konfigurácie automatizovanou formou

ID Identification – unikátny reťazec znakov na jednoznačnú identifikáciu

IO Input Output – označenie vstupno-výstupnej komunikácie

IP Internet Protocol – komunikačný protokol používaný medzi zariadeniami v sieti na výmenu paketov

JSON JavaScript Object Notation – štandardizovaný textový formát určený na prenos informácii, typicky medzi rôznymi prostrediami

NMS Network Management System – Systém na správu siete, zaužívaný názov pre riadiaci server v automatizačných nástrojoch

OS Operating System – operačný systém

SCP Secure Copy – šifrovaná forma protokolu na prenos dát

SDN Software defined network – softvérovo definovaná sieť, pokročilý model siete založený na centrálnej správe prostredníctvom kontroléra

SFTP Secure File Transfer Protocol – protokol na výmenu súborov zabezpečený použitím SSH tunela

SNMP Simple Network Management Protocol – jednoduchý aplikačný protokol používaný na správu zariadení, typicky v lokálnej sieti

SSH Secure Shell – aplikačný protokol používaný pre zabezpečené spojenie so vzdialeným zariadením

YAML Jednoduchý textový formát určený na zápis informácie, jednoducho čitateľný pre človeka

Zoznam príloh

A Usporiadanie modulov generátora

- `program` — koreňový priečinok nástroja
  - `file_io` — vstupno-výstupný (IO) modul
    - `__init__.py`
    - `ansible_dir.py`
    - `logger.py`
    - `vault_io.py`
    - `yaml_io.py`
  - `processing` — modul spracovania dát
    - `__init__py`
    - `arg_parser.py`
    - `configuration.py`
    - `data_processing.py`
    - `threadpool_py`
  - `rules` — modul pravidiel
    - `__init__py`
    - `dynamic_value.py`
    - `generator_rule.py`
    - `iterator_regex.py`
    - `list_reader.py`
    - `network.py`
    - `port_range.py`
    - `type_detector.py`
  - `test` — testovací modul
    - `app_test.py`
    - `unit_test.py`
  - `ans_gen.py` — hlavný zdrojový súbor generátora
  - `production_ans_gen.sh` — produkčné spustenie generátora

B Testovací skript pre automatizované vstupy

```
#!/usr/bin/expect
set cmd [lrange $argv 1 end]
set val [lindex $argv 0]
eval spawn $cmd
expect "template:"
send "$val\r";
interact
```

C Testovanie paralelizácie generovania konfigurácií

Tab. C.1: Časová náročnosť generovania výstupných konfigurácií použitím x vlákien

počet iterácií	1 vlákno	2 vlákna	4 vlákna	8 vlákien	16 vlákien	32 vlákien
1	0,651 s	0,643 s	0,655 s	0,652 s	0,660 s	0,654 s
5	0,760 s	0,744 s	0,756 s	0,763 s	0,748 s	0,752 s
10	0,856 s	0,851 s	0,870 s	0,879 s	0,889 s	0,893 s
25	1,230 s	1,268 s	1,250 s	1,264 s	1,281 s	1,386 s
50	1,790 s	1,892 s	1,947 s	1,930 s	1,996 s	2,062 s
100	3,090 s	3,192 s	3,282 s	3,261 s	3,305 s	3,431 s
250	6,423 s	6,685 s	7,160 s	7,234 s	7,421 s	7,562 s

D Vplyv šifrovania a dešifrovania na vykonávací čas

Tab. D.1: Časová náročnosť generovania šifrovaných výstupných konfigurácií použitím nešifrovanej šablóny použitím x vlákien

počet iterácií	1 vlákno	2 vlákna	4 vlákna	8 vlákien	16 vlákien	32 vlákien
1	0,723 s	0,726 s	0,733 s	0,734 s	0,733 s	0,731 s
5	0,997 s	0,940 s	0,921 s	0,912 s	0,923 s	0,930 s
10	1,342 s	1,241 s	1,140 s	1,129 s	1,138 s	1,192 s
25	2,381 s	1,973 s	1,798 s	1,812 s	1,872 s	1,896 s
50	4,172 s	3,142 s	2,843 s	2,872 s	2,942 s	3,018 s
100	7,619 s	5,672 s	4,972 s	4,989 s	5,053 s	5,180 s
250	18,024 s	12,759 s	11,284 s	11,190 s	11,325 s	11,478 s

Tab. D.2: Časová náročnosť generovania šifrovaných výstupných konfigurácií použitím šifrovanej šablóny použitím x vlákien

počet iterácií	1 vlákno	2 vlákna	4 vlákna	8 vlákien	16 vlákien	32 vlákien
1	0,736 s	0,729 s	0,739 s	0,735 s	0,733 s	0,742 s
5	1,130 s	0,992 s	0,982 s	0,991 s	0,983 s	1,002 s
10	1,712 s	1,381 s	1,349 s	1,297 s	1,298 s	1,287 s
25	3,195 s	2,358 s	2,231 s	2,227 s	2,240 s	2,231 s
50	6,132 s	3,962 s	3,852 s	3,782 s	3,779 s	3,814 s
100	11,479 s	7,113 s	7,073 s	6,821 s	6,872 s	6,892 s
250	27,590 s	17,412 s	16,309 s	16,375 s	16,689 s	17,245 s