VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SPRÁVA SERVEROVÝCH FAREM

CLUSTER MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DOMINIK HLAVÁČ ĎURÁN**
AUTHOR

VEDOUCÍ PRÁCE — **Mgr. ROMAN TRCHALÍK, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20799/2017/xhlava42

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Hlaváč Ďurán Dominik**
Obor: Informační technologie
Téma: **Správa serverových farem**
Cluster Management
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se nástrojem Ansible a jeho nasazením v OS RedHat
2. Navrhněte a vytvořte nástroj (sadu skriptů) pro evidenci HW (typy komponent, sériová čísla, verze firmware, apod.)
3. Navrhněte a vytvořte nástroj pro sledování zdrojů (CPU, RAM, disk I/O, síťová rozhraní, apod.). Výstupy pak znázorněte v grafu (např. MS Excel)
4. Navrhněte sadu skriptů pro běžnou správu, např. přenášení souborů, instalace aplikace, získání seznamu instalovaných programů, apod.
5. Proveďte vyhodnocení a srovnání s existujícími nástroji.

Literatura:

- Ansible Documentation. Red Hat, Inc. [online]. Rev. 2007-10-01 [cit. 2017-10-01]. Available at URL: < http://docs.ansible.com/>.
- Dále podle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Trchalík Roman, Mgr., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Bakalárska práca sa zaoberá návrhom systému pre komplexnú správu a monitorovanie serverových fariem založených na operačnom systéme *Red Hat Enterprise Linux/CentOS* pomocou nástroja *Ansible*. V práci je uvedený stručný popis použitých technológií a problematiky správy serverových fariem. Nami implementovaný nástroj zaisťuje prípravu prostredia pre použitie systému, vzdialenú inštaláciu a konfiguráciu aplikácií, registráciu hardware, monitorovanie softwarovej a hardwarovej funkcionality systému a mnoho iného. Následne je uvedené porovnanie výsledného systému s už existujúcimi nástrojmi. Súčasťou práce sú aj praktické ukážky riešenej problematiky.

Abstract

The bachelor thesis focuse on the design of a system for complex administration and monitoring server cluster based on the *Red Hat Enterprise Linux/CentOS* operating system using the *Ansible* tool. In the thesis there is a brief description of used technologies and issues of management of server cluster. The tool, that was implemented by us, provides preparation of the environment for system deployment, remote installation and configuration of applications, hardware registration, software and hardware system monitoring and much more. Consequently, there is a comparison of the resulting system with existing tools. Practical examples of the issue dealt with is also part of the thesis.

Kľúčové slová

cluster, Ansible, server, Linux, role, scenár, správa serverov, serverová farma

Keywords

cluster, Ansible, server, Linux, role, playbook, server managment, server farm

Citácia

HLAVÁČ ĎURÁN, Dominik. *Správa serverových farem.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Roman Trchalík, Ph.D.

Správa serverových farem

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Mgr. Romana Trchalíka, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Dominik Hlaváč Ďurán
23. mája 2018

Poďakovanie

Týmto by som sa chcel poďakovať vedúcemu práce Mgr. Romanovi Trchalíkovi, Ph.D. za odborné vedenie, rady a čas, ktorý mi pri tvorbe práce venoval. Ďalej by som sa chcel poďakovať pánovi Martinovi Homolovi z firmy Master Internet s.r.o.,ktorý mi s kolegami poskytol informácie o reálnom nasadení správy serverových fariem.

Obsah

Kapitola 1

Úvod

So zvyšujúcim sa množstvom aplikácií, služieb a dát, ktoré je potrebné uložiť alebo prevádzkovať, sa zvyšuje aj dopyt po rozširovaní a vzniku nových serverov. Tieto servery sú následne spájané do väčších celkov, ktoré tvoria clustery a ich spojením vznikajú serverové farmy. K spájaniu serverov dochádza z viacerých príčin ako napríklad: zvýšenie úložného priestoru, zvýšenie výpočetného výkonu, zaistenie vyššej dostupnosti alebo aj kvôli možnosti prevádzkovať viacero virtuálnych strojov na jednom fyzickom zariadení. Serverové farmy môžu tvoriť stovky až tisícky zariadení, pričom sa môžu líšiť v použitom hardware alebo účele použitia. Tento fakt zas ovplyvňuje formu správy clusterov alebo serverových fariem, kde sa so zvyšujúcim počtom zariadení, zvyšujú aj nároky na prostriedky na správu softwarových a hardwarových komponentov.

Na správu serverových fariem sa bežne používajú nástroje, ktoré umožňujú nasadzovanie a monitorovanie softwaru a hardwaru. Sú to napríklad nástroje ako: *Saltstack*, *Puppet*, *Chef* či *Foreman*. Všetky tieto nástroje majú väčšinou pomerne veľké požiadavky na systém, na ktorom môžu bežať, neposkytujú dostatočnú funkcionalitu alebo sú náročné na ovládanie.

Cieľom bakalárskej práce je návrh a implementácia systému na správu serverových fariem s jeho nasadením v operačných systémoch *Linux* špeciálne *RedHat Enterprise Linux*. Pri tvorbe systému bude využitý nástroj *Ansible*. Nástroj bude použit pri nasadzovaní, registrácií hardwaru a monitorovaní softwarovej a hardwarovej funkcionality. Na záver si uvedieme porovnanie s existujúcimi riešeniami.

V druhej kapitole sa zameriame na použité technológie a prostredie, ktoré sme pri práci použili, ale takisto aj na teoretické poznatky, ktoré nám pomôžu lepšie pochopiť problematiku správy serverových fariem. V tretej kapitole si predstavíme návrh systému a vo štvrtej kapitole sa pozrieme priamo na implementáciu daného návrhu. V poslednej piatej kapitole porovnáme výsledky implementovaného systému s už existujúcimi nástrojmi na správu serverových fariem.

Kapitola 2

Teória

V tejto kapitole si priblížime technológie použité pri implementácií a prostredie, ktoré sme použili. Takisto si predstavíme aj teoretické poznatky, ktoré nám pomôžu lepšie pochopiť problematiku správy serverových fariem.

V rámci prostredí sa zamerieame na rodinu *Unixových* operačných systémov a to presnejšie na *Linux* a jeho distribúcie *Red Hat Enterprise Linux* a *CentOS*.

2.1 Operačný systém

V literatúre pre pojem operačný systém nie je jednotná definícia. Niekde sa uvádza, že ho tvorí len jadro operačného systému niekde, že ho tvorí spolu s jadrom aj užívateľské rozhranie, systémové knižnice a ovládače.

Jeden funkčný celok tvorí jadro operačného systému a druhý užívateľské rozhranie spolu so systémovými knižnicami a ovládačmi.

Všeobecne sa dá povedať, že operačný systém je základná zložka programového vybavenia, ktorá prispôsobuje technické prostriedky stanoveným typom aplikácií a požadovanému režimu činnosti počítača.[1]

Operačný systém je to software, ktorý vytvára základnú spojujúcu medzivrstvu medzi užívateľom, hardwarom a užívateľskými aplikáciami. Inak povedané, operačný systém umožňuje užívateľom ovládať užívateľské aplikácie pomocou hardwarového vybavenia počítača. Zaisťuje predávanie správ medzi procesmi a umožňuje synchronizovať ich činnosť, poskytuje procesom systémové údaje.

Hlavnými úlohami operačného systému je:

- Správa prostriedkov systému - rozdeľovanie zdrojov systému (CPU, pamäť, disky, porty atď.) medzi procesy operačného systému a uživateľské aplikácie s dôrazom na ich najefektívnejšie využitie.
- Poskytnúť prostredie pre ktoré zabezpečuje užívateľovi intuitívny prístup k používateľským aplikáciam a vytvára nad nimi istú abstrakciu.

2.1.1 Jadro operačného systému

Jadro operačného systému alebo inak *kernel* je centrálny prvok väčšiny operačných systémov a zároveň najnižšia a najzákladnejšia časť systému, ktorá beží väčšinou v privilegovanom režime. Je zavádzaný do operačnej pamäte hneď pri štarte a beži počas celeho chodu systému.

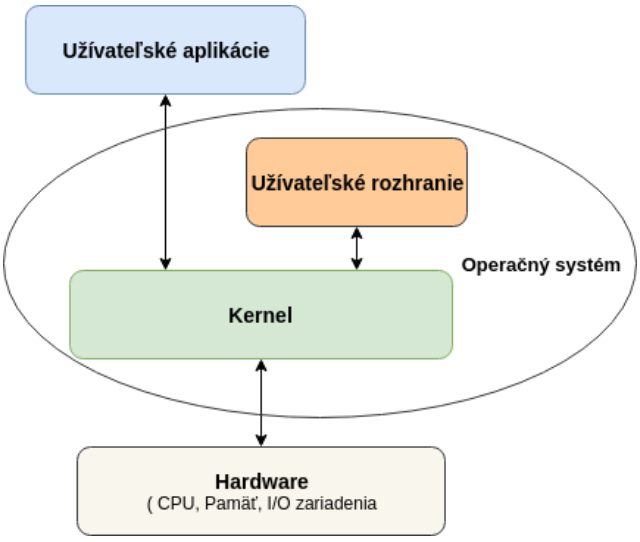
Kernel spravuje chod celého operačného systému, komunikuje so všetkými ovládačmi, správou pamäti a riadi systémové volania. Kernel vytvára procesy, prideľuje im pamäť a ostatné zdroje, nahráva programový kód do pamäte a spúšťa program.[2] Takisto spravuje a zapuzdruje komunikáciu s hardwarovými prvkami, spravuje sieťovú komunikáciu, multitasking a podobne.

2.1.2 Užívateľské rozhranie operačného systému

Užívateľské rozhranie je časť operačnáho systému pomocou ktorého užívateľ komunikuje s kernelom a aplikáciami operačného systému. Poskytuje užívateľovi prístup k abstrahovaným dátam z kernelu a taktiež mu umožňuje dáta zadávať.

Podľa formy interakcie užívateľské rozhranie pri v operačných systémoch vieme rozdeliť na 2 základne typy[3]:

- **Textové užívateľské rozhranie** - *TUI*. Zobrazuje počítačovú grafiku v textovej forme. V súčasnosti sa vo väčšine operačných systémov zachovalo ako samostatná aplikácia, v prípade *Linuxu* ide o *Shell*.
- **Grafické užívateľské rozhranie** - *GUI*. Je to rozhranie ktoré umožňuje ovládať systém a aplikácie pomocou grafických interaktívnych prvkov.



Obr. 2.1: Diagram zloženia operačného systému

2.2 Unix

Historia *Unixu* siaha do roku 1964, kedy bol založený projekt *Multics*. Projekt bol vytvorený za účelom vytvorenia komplexného operačného systému. Cieľom bolo vytvoriť výpočtový systém, ktorý by poskytoval výpočtové prostriedky pomocou vzdialeného prístupu.[4] Na projekte spolupracovali Bell Telephone Laboratories, MIT (Massachussets Institute of Technology) a General Electric Company.

Unix je rodina univerzálnych, viacužívateľských a viacúlohových operačných systémov. Medzi jeho nesporné výhody patrí prenositeľnosť kódu, bezpečnosť a stabilita.

Medzi Unixové systémy patrí:

- Systém V
- BSD - *Berkeley Software Distribution*
- Linux

2.3 GNU/Linux

GNU/Linux je popisovaný ako slobodný operačný systém, ktorý môže použiť hocikto.[5]. Slobodný v zmysle *Open Source* čiže je prístupný pre hocikoho a každý do neho môže prispievať alebo ho využívať. *GNU/Linux* vychádza z konceptu *Unixu* a patrí to rodiny takzvaných *Unix-like* operačných systémov. Je kompatibilný s väčšinou softwaru vyvíjaného pre Unix, ako napríklad *Bourne shell*, *X Window*, *mkfs*, *fsck* a mnoho iných. V súčastnej dobe je *GNU/Linux* vyvíjaný rozsiahlou komunitou vývojárov, ktorú tvorí množstvo veľkých firiem ako napríklad Intel, Red Hat, IBM, SUSE či mnoho iných, ale aj širokou škálou jednotlivcov, ktorý prispievajú k vývoju. Linuxový kernel je jeden z najväčších, najrýchlejšie sa vyvíjajúcich *Open Source* projektov v histórií technológie.[6]

Použitie operačného systému *GNU/Linux* je na vzostupe, a to hlavne pri použití na serveroch a superpočítačoch. Čo sa týka 500 najvýkonejších superpočítačov sveta tak má až 99.6% zastúpenie[7] Využitie *GNU/Linux* na osobných počítačoch je ťažšie merateľné, nakoľko ide o open source softwér, nedajú sa použiť metriky ako počet predaných alebo aktivovaných licencií, ktoré by určovali presný počet. Celkovo sa odhaduje percentuálne zastúpenie medzi osobnými počítačmi na 5-8 %, pričom je používaný hlavne vo vývojárskej komunite a medzi zástupcami akademickej obce.

Názov *GNU/Linux* je odvodený od dvoch samostatných projektov.

- GNU
- Linux

GNU Is Not Unix

GNU Is Not Unix (*GNU*) je projekt, ktorého vývoj započal v roku 1984 MIT pod vedením Richarda Stallmana na MIT. Hlavnou myšlienkou bolo vytvoriť operačný systém, ktorý bude kompatibilný s Unixom, bude poskytovať užívateľom slobodu a umožňuje im spúšťať, kopírovať, distribuovať, študovať a vylepšovať softvér podľa vlastných potrieb.

Linux

Linuxové jadro bolo vyvinuté Linusom Torvaldsom v roku 1991. Hlavnou príčinou vzniku bolo to, že v tej dobe dostupné kernely operačných systémov neboli dostačujúce pre potreby Torvaldsona. Zdrojové kódy projektu *BSD* neboli úplne verejné dostupné a vývoj GNU kernelu *HURD* uviazol a jeho vývoj by trval ďalšie roky.[5]. Do Linuxového kernelu boli integrované viaceré nástroje z *GNU*, aj vďaka čomu kernel *HURD* postupne nahradil Linuxový kernel.

Red Hat Enterprise Linux a CentOS

Red Hat Enterprise Linux (RHEL) je Linuxovo založený operačný systém vyvíjaný spoločnosťou *Red Hat* na komerčné účely čo zahŕňa hlavne podporu. Medzi jeho hlavné pred-

nosti patrí stabilita, vysoká miera bezpečnosti, spoľahlivosť, Linuxove kontajnere a mnoho iného.[8] *RHEL* patrí medzi svetovo najrozšírenejšie podnikové Linuxové distribúcie.

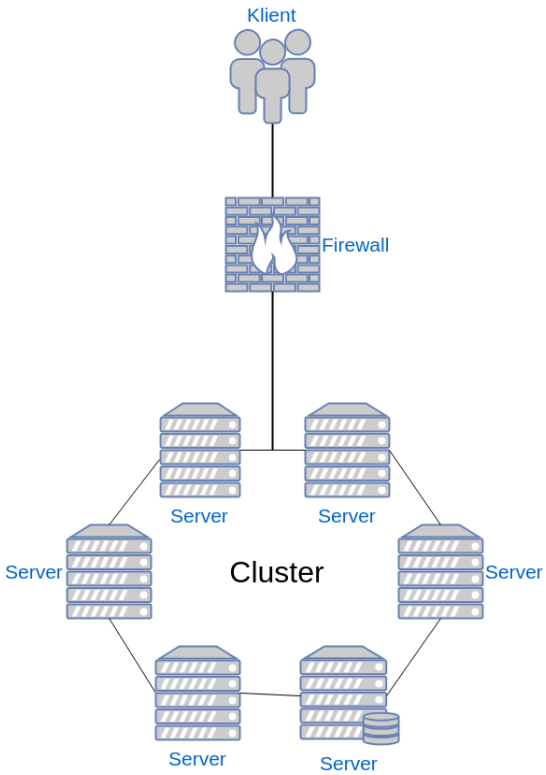
CentOS je operačný systém odvodený z *Red Hat Enterprise Linuxu* , je s ním plne kompatibilný a založený na zdrojových kódoch z RHEL avšak stará sa o neho komunita a je voľne dostupný pre všetkých užívateľov. Tak ako *CentOS* aj niektoré ďalšie distribúcie vychádzajú z *RHEL* a sú to:

- *Oracle Enterprise Linux*
- *Scientific Linux*
- *Pie Box Enterprise Linux*

2.4 Cluster

Nároky na výkon, úložný priestor alebo dotupnosť serverov stále rastú, pričom výkonnejší hardware je častokrát príliš drahý. Slabší hardware by zas výpočetne náročné úlohy nezvládal alebo neposkytoval dostatočné miesto pre uloženie dát. Riešením je preto spájanie viacerých serverov do clustrov, ktoré ponúkajú dobrý pomer ceny a výkonu. Medzi najčastejšie dôvody vzniku počítačových clustrov patrí zvýšenie výpočetného výkonu, zväčšenie úložného priestoru, zvýšenie spoľahlivosti alebo prerozdeľovanie záťaže.

Počítačový cluster je definovaný ako zoskupenie aspoň dvoch počítačov (uzlov), ktoré spolu spolupracujú a navonok sa tvária ako jeden počítač[9].

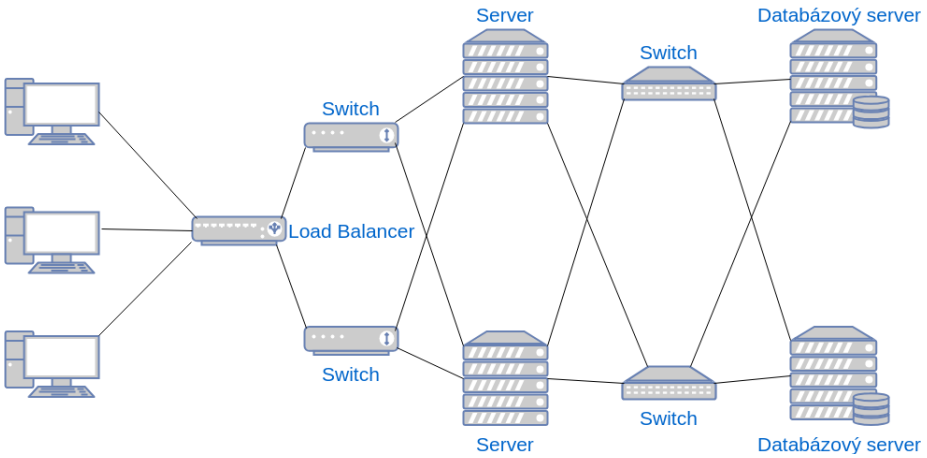


Obr. 2.2: Príklad schémy Clusteru

Typy clustrov

Rozlišujeme 4 základné typy clustrov:[10]

- **Úložný cluster** (*Storage cluster*) - je špeciálny typ clusteru s rozložením záťaže, pričom poskytuje konizstentný obraz súborového systému naprieč servermi v clustri, kde umožňuje simultálny zápis alebo čítanie zdieľaných súborov. Uľahčuje administráciu, zálohovanie a zotavovanie sa z chýb pri práci so súborovým systémom. Tým, že vytvára jeden súborový systém, umožňuje jednoduhšiu inštaláciu a aktualizáciu. Používa sa špeciálny súborový sýstém, v prípade *RHEL* je to *GFS* taktiež od spoločnosti Red Hat. Zaisťuje konzistenciu, integritu dát, redundanciu, mechanizmus zamykania súborov a pokrytie výpadkov systému.
- **Cluster s vysokou dostupnosťou** (*High availability cluster*) - v dôsledku používania, aktualizácii systému alebo aplikácia môže dôjsť k chybe a výpadku služieb. Clustre s vysokou dostupnosťou zabezpečujú nepretržitú dostupnosť služieb odstránením bodov zlyhania a zabezpečením služieb z iného bodu clustra v prípade, že primárny uzol prestane fungovať. Zvyčajne služby bežiace v clustri s vysokou dostupnosťou používajú zdieľaný súborový systém kvôli tomu, že v prípade výpadku musí mať náhradny uzol prístup k údajom prímárneho uzla. Preto musí cluster s vysokou dostupnosťou udržiavať integritu dát. Zlyhania uzlov v rámci clustra s vysokou dostupnosťou nie sú viditeľné z klientov mimo clustra, vďaka čomu umožňuje tento koncept taktiež vykonávať plánovanú údržbu bez toho, aby používateľ pocítil výpadok alebo zníženie výkonu.



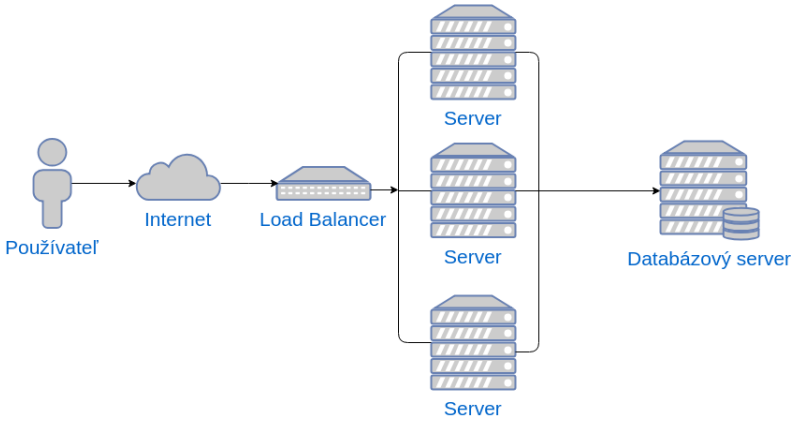
Obr. 2.3: Cluster s vysokou dostupnosťou

Spôsoby vysporiadania sa so zlyhaním:[11]

  - **Studené zlyhanie** - je metóda, pri ktorej je identický systém v zálohe a je spustený len v prípade zlyhania primárneho systému.
  - **Horúce zlyhanie** - je redundantná metóda, pri ktorej beží sekundárny systém simultálne s primárnym a v prípade porúch, sekundárny systém prevezme úlohy primárneho systému.

- **Cluster s rozložením záťaže** (*Load balancing cluster*) - rozosiela požiadavky sieťových služieb na viac uzlov clusteru za účelom rozloženia záťaže. Rozloženie záťaže

poskytuje nákladovo efektívnu škálovateľnosť vďaka tomu, že sa dá prispôsobovať počet uzlov podľa aktuálnej záťaže. Princíp fungovania clustra s rozložením záťaže je, že je určený primárny server tzv. *Master load balancer* a ďalšie sekundárne servery. Ak primárny server v clusteri prestane fungovať, softvér na rozloženie záťaže rozpozná chyby a presmeruje požiadavky na ďalšie uzly clustra.[9] Typicky sa tento typ clustrov často používa pre webové servery, kde zaisťuje vysokú rýchlosť a zároveň garanciu dostupnosti služieb.



Obr. 2.4: Cluster s rozložením záťaže

Spôsoby rozloženia záťaže:[11]

- **Rozloženie záťaže pomocou DNS** - funguje tak že služba DNS rozkladá záťaž medzi viaceré uzly. Nevýhodou je, že nie je možné zvoliť algoritmus pre rozdeľovanie záťaže, ale vždy sa používa algoritmus Round Robin [1].
- **Rozloženie záťaže pomocou hardware** - používa dedikovanné hardwarové zariadenia k rozloženiu záťaže prevádzky uzly clusteru.
- **Rozloženie záťaže pomocou software** - je to jedna z najspoľahlivejších metód rozdelenia záťaže. Používajú sa rôzne špecializované algoritmy k rozdeleniu záťaže prevádzky medzi jednotlivé uzly clusteru.

- **Cluster s vysokým výkonom** (*High performance cluster*) -

  využíva sa pri náročných výpočtoch, kde by bolo použitie výkoného serveru príliš drahé. Clustre s vysokým výkonom používajú uzly clusteru na vykonávanie súbežných výpočtov, vďaka čomu dosiahnu potrebný výpočetný výkon. takýto cluster sa skladá z aspoň dvoch počítačov, ktoré sú prepojené vysokorýchlostnou sieťou a zdieľajú medzi sebou výpočetné zdroje.

Správa clustrov

Správa clustrov zahŕňa dve základne činnosti:

- **Správa hardwarového vybavenia** - zahŕňa registráciu hardwaru, monitorovanie funkčnosti jednotlivých hardwarových komponentov, ako napríklad monitorovanie výkonu a zaťaženia jednotlivých komponentov serverov, zaplnenie diskov, využitie operačnej pamäte, stav sieťových zariadení.

[1] https://t4tutorials.com/round-robin-process-scheduling-algorithm-in-operating-systems/

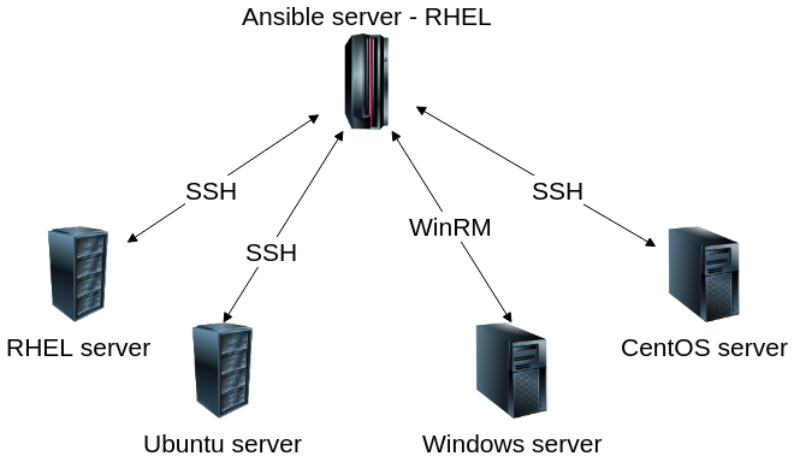
- **Správa softwarového vybavenia** - zahŕňa inštaláciu operačného systému, inštaláciu a konfiguráciu softwarového vybavenia napríklad: webový server, databázový server, vyrovnávač záťaže, ale aj pridávanie a správa užívateľov a súborov v systéme, zálohovanie dát a databázy, monitorovanie funkčnosti inštalovaného softwaru. Vždy v závislosti na účele použitia.

2.5 Ansible

Ansible je open source nástroj na automatizáciu nasadzovania software, správu konfigurácie, orchestráciu, sieťovú automatizáciu a mnoho iného.[12]

Ansible funguje na princípe klient-server pričom jeho nespornou výhodou je, že k svojmu fungovaniu nepotrebuje predinštalovanie agentov na spravovaných uzloch, stačí ak je nainštalovaný programovací jazyk Python[2] vo verzii 2.6 alebo vyššej.[13]. V kontexte používania Ansible sa jednotlivé servery alebo zariadenia s ktorými sa pracuje volajú uzly (*nodes*). Pri komunikácií sa využíva protokol SSH (*Secure Shell*), ktorý slúži k šifrovaniu komunikácie, pokiaľ ide o komunikáciu so zariadeniami, ktoré bežia na operačnom systéme *Windows*, je možné použiť protokol *WinRM*.

Ansible funguje na princípe scenárov a rolí scenárov, ktoré popisujú jednotlivé úlohy, ktoré sa majú na spravovaných zariadeniach vykonať. Výstup jednotlivých scenárov je v formáte *JSON*, kde je zobrazený celý priebeh s informáciami o behu scenárov, zmenených súboroch a jednotlivých úlohách scenárov.



Obr. 2.5: Schéma sieťe pri použití Ansible

[2] https://www.python.org/

Formáty súborov používane nástrojom Ansible

Ansible môže využívať rôzne typy fomátov súborov k správe uzlov, avšak medzi základné patria 3 typy formátov súborov:

- *YAML*[3] (*Ain't Markup Language*) je štrukturovaný formát slúžiaci na serializáciu dát vo forme, ktorá je pre ľudí ľahko čitateľná.
- *INI*[4] je jednoduchý textový formát zložený z častí vlastností a hodnôt.
- *JSON*[5] (*JavaScript Object Notation*) je odľahčený formát pre výmenu dát.

Inventár

Inventár alebo v originále *Inventory* je súbor, ktorý obsahuje zoznami uzlov s ktorými Ansible pracuje. Uzly môžu byť rozdelené do skupín, ktoré majú niektoré vlastnosti rovnaké, a taktiež jedna skupina môže obsahovať sama aj iné skupiny. Existujú dve typy inventárov.

- **Statický Inventár** - môže byť reprezentovaný formou jednoduchého textového súboru vo formáte *INI* alebo vo formáte *YAML*. Obsahuje záznami o jednotlivých uzloch a taktiež môže obsahovať aj ďalšie informácie o danných uzloch, ako napríklad premenné.
- **Dynamický Inventár** - je reprezentovaný formou skriptu v ľubovoľnom programovacom jazyku, ktorý dynamicky generuje informácie o skupinách, uzloch a premenných vo formáte *JSON*. Jedinnou podmienkou pri tvorbe skriptu je aby ho bolo možné spúšťať s parametrom `--list`, ktorý vytlačí na štandardný výstup dáta vo formáte *JSON*.

Dynamický inventár je vhodné používať pri veľkom množstve uzlov alebo ak uzly alebo skupiny obsahujú väčšie množstvo premenných. Na druhú stranu ak je potrebné zašifrovať dáta, ako napríklad prístupové údaje užívateľov k uzlom, je dobre použiť statický inventár, kde je možné dáta šifrovať pomocou *Ansible Vault*.

Inventár môže taktiež obsahovať priečinky `host_vars` alebo `group_vars`, ktoré obsahujú informácie platné pre danné uzly alebo skupiny uzlov.

```
test.example.com

[webservers]
nodejs1.example.com
nodejs2.example.com

[dbservers]
mysql-east.example.com
mysql-west.example.com
```

Výpis kódu 2.1: Príklad inventory súboru vo formáte INI

[3] http://yaml.org/spec/1.2/spec.html
[4] https://www.pcmag.com/encyclopedia/term/44993/ini-file
[5] https://tools.ietf.org/html/rfc7159

```yaml
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: Ensure that apachee is at the latest version
    yum:
      name: httpd
      state: latest
  - name: Copy the apachee config file
    template:
      src: /src/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - Restart apachee
  - name: Ensure apache is running
    service:
      name: httpd
      state: started
```

Výpis kódu 2.2: Príklad scenára vo formáte YAML

Modul

Moduly v Ansible sú základné jednotky na vykonávanie a konfiguráciu úloh. Moduly predstavujú skripty, ktoré môžu byť vykonávané v režime ad-hoc[14] alebo môžu byť zoradené do zoznamov, ktoré sa nazývajú scenáre alebo v originále *Playbook*.

Ansible poskytuje širokú základňu modulov, ktoré poskrývaj najrôznejšie spektrum potrieb pri používaní nástroja *Ansible* na správu,nasadzovanie alebo konfiguráciu serverov. Moduly je možné si implementovať aj sám, a to v rôznych programovacích jazykoch.

Moduly uľahčujú používanie nástroja *Ansible*, nakoľko poskytujú priamy prístup k niektorým funkciám, ako napríklad modul `yum` uľahčuje pracovanie s repozitármi a nie je nutné používať ad-hoc príkazy.

Scenár

Scenár predstavuje postupnosť prevedenia príkazov z jednotlivých modulov alebo ad-hoc príkazov, ktoré sa majú vykonať na zariadeniach uvedených v inventári[15]. Scenár je súbor napísaný vo formáte *YAML*, čo zabezpečuje jeho dobrú čitateľnosť. Scenár môže obsahovať viacero úloh, ktoré sa majú vykonať a je možné pre neho definovať rôzne premenné, ktoré sa použijú pri jednotlivých uzloch. Serverové farmy obsahujú veľké množstvo rozličných druhov serverov, ako napríklad databázové alebo webové servery. Ansible scenár, ktorý by mal zabezpečovať konfiguráciu a managment takého množstva serverov, by sa stával postupne nepriehľadný. Preto sa kvôli znovupoužiteľnosti a väčšej variabilite ďalej škálujú na role.

Rola

Rola predstavuje oddelený logický celok, ktorý delí vykonanie špecifickej úlohy na menšie podúlohy, ktoré je možne opakovať a kombinovať s inými rolami. Umožňuje použitie ďalších skriptov, konfiguračných súborov alebo premených špecifických pre dannú úlohu. Táto funkcionalita umožňuje efektívne využívať role a zabraňuje duplicite kódu nakoľko, napríklad pokiaľ by sme chceli nainštalovať webové servery s rôznymi databázami, stačí nám to definovať v scenári a nie je potrebné písať variácie scenárov pre každú kombináciu aplikácií.

```
---
- name: Check if EPEL repo is configured
  stat:
    path: "/etc/yum.repos.d/epel.repo"
  register: epel_repofile_result

- name: Install EPEL repo when not configured
  yum:
    name: epel
    state: present
  register: result
  when: not epel_repofile_result.stat.exists
```

Výpis kódu 2.3: Príklad role vo formáte YAML

Rola je reprezentovaná adresárom, ktorý obsahuje ďalšie podadresáre, ktoré obsahujú buď súbory vo formáte *YAML* ktorých sú postupnosti príkazov, ktoré sa majú vykonať alebo obsahujú premenné špecifické pre dannú úlohu. Základným súborom, ktorý sa spúšťa v roli alebo obsahujé informácie je `main.yml`, ďalšie súbory môžu byť definované v ňom.

```
common/ # Hierarchy represents a "role"
      tasks/ #
          main.yml # <-- tasks file can include smaller files if warranted
      handlers/ #
          main.yml # <-- handlers file
      templates/ # <-- files for use with the template resource
          tmp.conf.j2 # <------- templates end in .j2
      files/ #
          bar.txt # <-- files for use with the copy resource
          foo.sh # <-- script files for use with the script resource
      vars/ #
          main.yml # <-- variables associated with this role
      defaults/ #
          main.yml # <-- default lower priority variables for this role
      meta/ #
          main.yml # <-- role dependencies
      library/ # roles can also include custom modules
      module_utils/ # roles can also include custom module_utils
```

Výpis kódu 2.4: Štruktúra adresára role

Zvýšenie privilégií

Zvýšenie privilégií alebo v originále *Privilege escalation* je funkcionalita, ktorá umožňuje užívateľovi pri spúšťaní úloh vystupovať ako iný užívateľ, napríklad ako používateľ, ktorý má administrátorske práva. Zvyšovanie privilégií je možné nastaviť na všetky spúšťané úlohy alebo aj jednotlivo, a taktiež je možné určiť užívateľa, pod ktorým sa budú úlohy spúšťať v režime zvýšenia privilégií.

Ansible Vault

Ansible Vault je nástroj, ktorý poskytuje *Ansible* a slúži na šifrovanie citlivých informácií. Tieto informácie môžu byť následne použité pri spúšťaní scenárov alebo v inventároch. Od verzie 2.4 je možné šifrovať súbory viacerými kľúčmi, čo znamená že jeden súbor môže byť zašifrovaný pomocou kľúča pre testovanie ale aj pomocou kľúča pre nasadenie, čo zvyšuje bezpečnosť vďaka škálovateľností použitia rôznych hesiel pre rôzne stupňe ochrany.

V hlavičké šifrovaného súboru sa nachádzajú informácie ako: identifikačné číslo vault súboru, verziu Vault software a skratku algoritmu, pomocou ktorého boli dáta zašifrované.

Ansible Tower

Ansible Tower je platená služba od spoločnosti Red Hat, ktorá k *Ansible* pridáva grafické rozhranie, rozširuje funkcionalitu o automatické spúšťanie skriptov, sledovanie štatistík v reálnom čase a mnoho iného. Medzi hlavné výhody patrí reťazenie scenárov a ich dynamcické spájanie na základe výsledkov predošlích úloh. Ďalej umožnuje prevádzať automatické testovanie, ktoré zahŕňa prípravu prostredia vrátanie inštalácie, nasadenie testovaného software, jeho automatické sustenie, vyhodnotenie výsledkov a odoslanie správ o výsledkoch. [16]

Kapitola 3

Analýza a návrh

V tejto kapitole sa budeme zaoberať návrhom systému na správu serverových fariem. Nakoľko je správa serverových fariem pomerne rozsiahla téma a dosť závislá na účele použitia jednotlivých serverov, tak sa pozrieme na dáta a informácie z reálneho nasadenia, ktoré sme získali v našom prieskume. Následne si predstavíme koncept návrhu implementácie, jednotlivé moduly systému a problematiku návrhu niektorých častí systému.

3.1 Analýza požiadavkov systému na správu serverových fariem

Správa serverových fariem je pomerne rozsiahla problematika a vcelku závislá na účele použitia daných serverov, preto je pomerne ťažké úplne zovšeobecniť potreby pri implementácií systému na správu serverových fariem. Za účelom získania detailnejších informácií a praktických poznatkov o správe serverových fariem sme sa rozhodli vykonať prieskum. Prieskum sme uskutočňovali formou dotazníka[1] či priamo návštevou dátových centier a konzultáciou so správcami serverových fariem.

Dotazník zodpovedalo celkom 13 respondentov, ktorí boli zložení zo správcov sietí a serverových clustrov pracujúcich v rôznych firmách. Na zhotovenie dotazníku sme využívali voľne dostupú službu *Google Forms*[2]. Dotazník obsahoval 5 sekcií a celkovo 14 povinných otázok a jedna nepovinná. Väčšina otázok mala formu výberu viacerých odpovedí s možnosťou doplnenia vlastnej odpovede.

Otázky boli rozdelené do 4 kategórií:

- **Software na správu serverových clustrov** - otázky sa týkali aktuálne používaného software na správu, výhod a nevýhod aktuálne používaných riešení a požadovanej funkcionality na systém na správu serverových friem.
- **Nasadzovanie software v serverových clustroch** - otázky sa týkali inštalácie aplikácií a úkonov potrebných automatizovať pri nasadzovaní a konfigurácií softwaru na serverových clustroch.
- **Monitorovanie serverových clustrov** - otázky sa týkali nástrojov používaných respondentmi pri monitorovaní serverových clustrov, požadovanej funkcionality, parametrov monitorovania a požadovaného formátu na uchovávanie dát.

[1] https://goo.gl/forms/pWvH28zh0J8dxSTs2
[2] https://www.google.com/forms/about/

- **Registrácia hardware v serverových clustroch** - otázky sa týkali toho aké informácie je požadované o hardware uchovávať, a taktiež v akom formáte ich uchovávať.

Interview sa konali so zamestnancami firmy MasterDC Brno a Palatine. So zástupcami firiem sme absolvovali prehliadku dátových centier, kde nám vysvetlili aké technológie používajú na správu a aké úkony je najčastejšie potrebné pri správe automatizovať. Ďalej sme konzultovali možnosti pri návrhu systému.

Účelom prieskumu bolo zistenie požiadaviek na implementáciu jednotlivých častí systému na správu serverových fariem, výberu najvhodnejšieho prístupu a zistenia najčastejšie používaných aplikácií v serverových clustroch. Prieskum splnil požadovaný účel a získali sme množstvo informácií, ktoré sme ďalej spracovali. Výstupom získaných informácií či už spôsobom dotazníka alebo osobných stretnutí, je zoznam požiadaviek a následný návrh jednotlivých modulov systému o ktorom sa viac dočítame v kapitole 3.3.

3.2 Požiadavky na vlastnosti systému

Poznatky získané z prieskumu či formou interview alebo dotazníka sme zhrnuli do nasledujúceho zoznamu požiadaviek. Pri každom z požiadavkov je uvedená priorita a odhadovaná náročnosť implementácie na stupnici od 1 - 5, kde 5 je najnáročnejší a 1 je najmenej náročný na implementáciu.

3.2.1 Vlastnosti systému

Požiadavok 1: modularita
Systém by mal poskytovať čo najväčšú modularitu, to znamená že by mali byť jednotlivé časti rozdelené do logických celkov a jednotlivé celky použiteľné nezávisle na sebe.
Priorita: vysoká
Odhadovaná náročnosť: 2

Požiadavok 2: jednoduchá správa
Systém by malo byť možné ovládať bez nutnosti učiť sa nový programovací jazyk alebo bez nutnosti vedieť programovať všeobecne.
Priorita: vysoká
Odhadovaná náročnosť: 4

Požiadavok 3: správa viacerých zariadení naraz
Systém by mal umožňovať spravovať viaceré zariadenia naraz
Priorita: stredná
Odhadovaná náročnosť: 1

Požiadavok 4: ľahká rozšíriteľnosť funkcionality
Systém by mal umožňovať rozšíriť funkcionalitu napríklad o podporu ďalších nástrojov alebo operačných systémov bez nutnosti veľkých úprav systému.
Priorita: stredná
Odhadovaná náročnosť: 2

Požiadavok 5: možnosť používať ako zdroj dát štrukturované súbory
Systém by mal umožňovať spracovávať dáta o spravovaných uzloch zo štrukturovaných formátov ako napríklad *CSV* alebo *JSON*.

Priorita: vysoká
Odhadovaná náročnosť: 3

Požiadavok 6: textové rozhranie
Systém by mal obsahovať textové užívateľské rozhranie.
Priorita: vysoká
Odhadovaná náročnosť: 3

Požiadavok 7: grafické rozhranie
Systém by mal obsahovať grafické užívateľské rozhranie.
Priorita: nízka
Odhadovaná náročnosť: 5

Požiadavok 8: zálohovanie súborov a databázy
Systém by mal byť poskytovať možnosť zálohovať súbory a databáze.
Priorita: stredná
Odhadovaná náročnosť: 4

3.2.2 Nasadzovanie software

Požiadavok 9: správa systémových prvkov
Systém by mal umožňovať správu systémových prvkov ako napríklad užívateľov, skupín alebo prácu so súborami.
Priorita: vysoká
Odhadovaná náročnosť: 1

Požiadavok 10: nasadzovanie a konfigurácia databázy
Systém by mal umožňovať nasadzovanie a konfiguráciu databázových systémov.
Priorita: vysoká
Odhadovaná náročnosť: 1

3.2.3 Registrácia hardware

Požiadavok 11: uchovávanie informácií
Systém by mal umožňovať uchovávať informácie o hardwarovom vybavení.
Priorita: vysoká
Odhadovaná náročnosť: 4

Požiadavok 12: súhrn informácií o hardware
Systém by mal poskytovať súhrné informácie o zastúpení a podiele hardwarových komponentov na spravovaných strojoch, napríklad formou grafov.
Priorita: vysoká
Odhadovaná náročnosť: 4

3.2.4 Monitorovanie

Požiadavok 13: monitorovanie hardwaru
Systém by mal umožňovať monitorovanie stavu hardwarových komponentov ako vyťaženosť CPU, teplota CPU, využitie diskov a podobne.
Priorita: vysoká
Odhadovaná náročnosť: 4

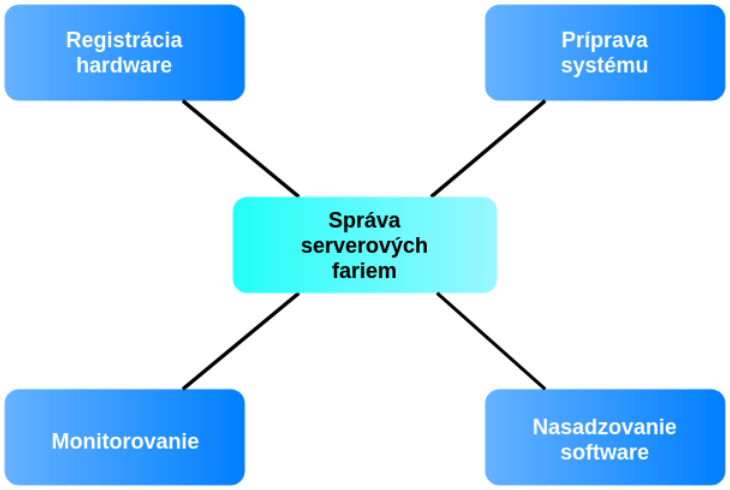
Požiadavok 14: monitorovanie softwaru
Systém by mal umožňovať monitorovanie stavu systémových služieb a aplikácií.
Priorita: vysoká
Odhadovaná náročnosť: 3

Požiadavok 15: prehľad monitorovania
Systém by mal poskytovať súhrné informácie o stave systému v čase napríklad formou grafov.
Priorita: stredná
Odhadovaná náročnosť: 3

3.3 Moduly systému

Aby sme splnili požiadavky na výsledný systém ktoré sme si stanovili v predošlej podkapitole 3.2, rozdelili sme si systém na moduly, ktoré predstavujú ucelené logické celky funkcionality. Každý z modulov predstavuje jednú funkčnú časť systému na správu serverových fariem. V rámci každého modulu bude priložený súbor `README.md` v ktorom bude popísaná funkcionalita, spôsob používania daného modulu a príklady použitia respektíve referencia na súbory `README.md` v menších logických celkoch systému.

Každý z modulov bude možné použiť aj samostatne, vďaka čomu bude možné kombinovať použitie jednotlivých modulov s inými nástrojmi bez nutnosti inštalovania nadbytočných modulov.



Obr. 3.1: Schéma návrhu modulov systému na správu serverových fariem

Príprava systému

Prípravou systému sa rozumie pripravenie systému na používanie bez ďalšej nutnosti doinštalovania software alebo nutnosti úprav. Operačné systémy *RHEL* a *CentOS* obsahujú predinštalovaný programovací jazyk Python, vďaka čomu môžeme využívať nástroj *Ansible*. *Ansible* sa doinštaluje pomocou jednoduchého postupu, ktorý bude priložený v súbore `README.md` modulu prípravy systému.

V rámci prípravy systému pre použitie systému je nutné previesť dáta o spravovaných systémoch a prístupových udajoch k nim. To znamená, že je nutné spracovať dáta do formy *Inventory* súborov a zabezpečiť citlivé informácie. Z prieskumu vyplynulo, že medzi najideálnejšie vstupné formáty pre spracovanie údajov o uzloch patrí CSV.

K prevodu dát z formátu *CSV* pripravíme skripty napísané v jazyku *Python*, vďaka čomu nepotrebujeme doinštalovať ďalší software, nakoľko *Ansible* sám využíva jazyk *Python*. Vytvoríme dva druhy skriptov, vďaka čomu budeme schopný previesť dáta do formy statického aj dynamického inventára. O zabezpečení dát si viac povieme v podkapitole 3.4.

Ako sme si už skôr spomenuli v kapitole 2.5, *Ansible* je nástroj, ktorý funguje na princípe klient - server a pri komunikácií používa protokol *SSH*. Natívnou súčasťou *Ansible* je aj používanie *SSH kľúčov*[3] pri komunikácií, preto implementujeme scenár, v ktorom automatizujeme nasadzovanie *SSH kľúčov* na spravované uzly.

Nasadzovanie a konfigurácia software

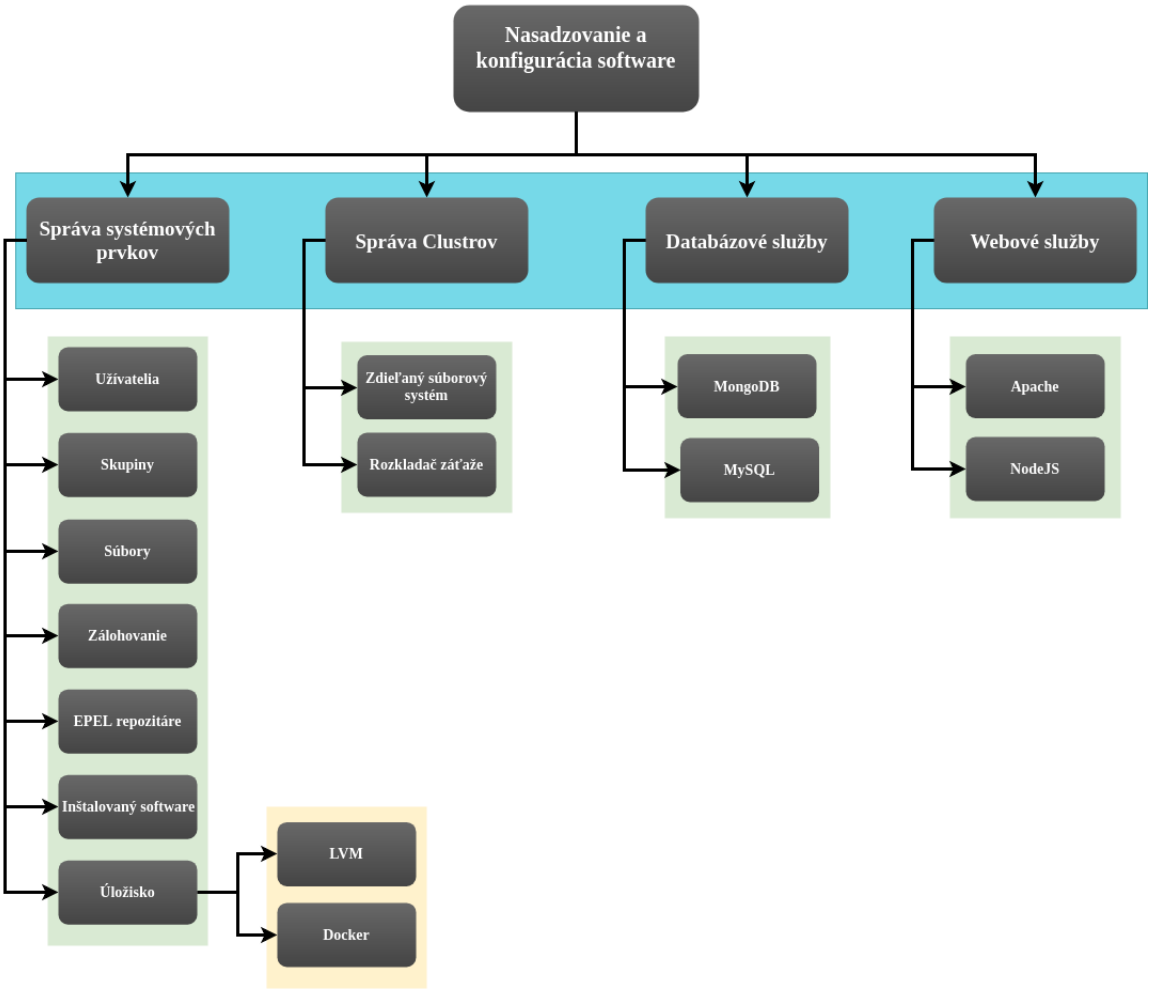
Na základe výsledkov prieskumu sme zistili informácie o tom, aký druh softwaru je najčastejšie potrebné pri príprave serverových clustrov nainštalovať a nakonfigurovať, respektíve čo patrí medzi najbežnejšie úkony pri správe, ktoré je potrebné automatizovať.

Vybrali sme najčastejšie úkony a rozdelili ich do menších logických celkov :

- Nasadzovanie a konfigurácia databázových systémov
- Nasadzovanie a konfigurácia webových služieb
- Príprava clustrov (prepojenie súborových systémov, príprava rozdeľovača záťaže)
- Správa systemových prvkov (práca s užívateľmi, práca so súbormi, zálohovanie atď.)

Detailnejší návrh rozdelenia modulu na menšie logické celky môžete vidieť na obrázku 3.2

[3] https://wiki.archlinux.org/index.php/SSH_keys



Obr. 3.2: Schéma návrhu rozdelenia modulu na nasadzovanie a konfiguráciu software

Následne pripravíme viacero scenárov s kombináciou často používaných úloh pri nasadzovaní a konfigurácií softwaru.

Registrácia hardware

V rámci správy clustrov je potrebné uchovávať informácie o používanom hardware z viacerých dôvodov, či už v prípade potreby výmeny za nové komponenty z dôvodu poruchy alebo upgradu hardwarových častí, a taktiež za účelom plánovania rozširovania clustrov o nové servery a zaisťenia ich kompatibility.

Podľa prieskumu, ktorý sme rozobrali v kapitole 3.1, sme zistili bližšie informácie, ktoré sú pre administrátorov alebo návrhárov systémov dôležité pri registrácií hardwaru serverov. Tieto dáta plánujeme získavať pomocou funkcie *Ansible* na zbieranie skutočností alebo v originále *Gathering Facts* a ich následne interpretovanie do formátu *CSV* pomocou skriptu napísaného v jazyku *Python*. Pri analýze sme taktiež zistili, že je vhodnejšie dáta spracovávať priamo na uzle na správu ako na jednotlivých spravovaných uzloch, a to za účelom lepšej kombinácie údajov, ktoré budeme potrebovať spracovať.

Monitorovanie

Monitorovanie je dôležitá súčasť systému, nakoľko nám poskytuje informácie o stave služieb, ktoré bežia alebo aj o stave hardwarového vybavenia.

V rámci monitorovania implementujeme scenáre, ktoré budú zahŕňať:

- **Monitorovanie software** (zistenie dostupnosti systému, kontrola funkcionality inštalovaných služieb atď.)
- **Monitorovanie hardware** (zbieranie dát o zaplnení diskov, stave CPU atď.)

Pri monitorovaní hardware využijeme nástroje ako *inxi*[4] alebo *top*[5], ktoré sú súčasťou buď priamo operačného systému *RHEL* a *CentOS* alebo *EPEL* repozitárov týchto systémov.

Monitorovanie je potrebné vykonávať v pravidelných intervaloch, aby sme mohli včas zachytiť prípadne problémy a vyriešiť ich, preto plánujeme pripraviť *CRON JOB*[6], ktorý bude spúšťať jednotlivé scenáre vykonavajúce monitorovanie systémov a ukladať dáta o jednotlivých systémoch.

3.4 Zabezpečenie

Takmer každá spoločnosť potrebuje uchovávať citlive alebo interné dáta a zabrániť ich úniku ku konkurencii či ochrániť vlastné prístroje a prístup k ním pred znuežitím nepovolanými osobami. Touto problematikou sa zaoberá *Prevencia Straty Dát* alebo v originále *Data Loss Prevention* (DLP).

Systém na prevenciu straty dát je návrh, ktorý rieši kontrolu obmedzení toho ako je s dátami pracované.[17] V týchto systémoch sa definujú pravidlá ako je s citivými dátami narábané a uplatňujú sa na ne špeciálne obmedzenia ako napríklad prístup len určitých užívateľov k súborom s dátami alebo viacfaktorové zabezpečenie prístupových údajov k dátam.

V našom prípade je nutné pri implementácií zabezpečiť ochranu citlivých dát ako napríkald prístupové údaje k spravovaným uzlom alebo heslá k používateľským aplikáciam, ktoré bežia na jednotilvých systémoch. Pre zabezpečenie citlivých dát v našom prípade použijeme rozšírenie nástroja *Ansible* a to *Ansible Vault*, ktorý sme si predstavili v kapitole 2.5.

3.5 Spracovanie a reprezentacia dát

Získané dáta je potrebné zjednotit do jedného formátu a následne pripraviť formu ich reprezentácie pre používateľov. Na základe informácií z prieskumu sme zisitili, že medzi najpolulárnejšie formáty pre reprezentáciu dát patrí formát *JSON* alebo *CSV*[7]. Nakoľko *Ansible* defaultne využíva reprezentáciu dát vo formáte *JSON* dáta plánujeme v tomto formáte aj ukladať a ďalej plánujeme implemenotavať spracovanie dát do formátu *CSV*. Na spracovanie dát využijeme skripty napísané v programovacom jazyku *Python*, následne

[4] https://github.com/smxi/inxi
[5] http://man7.org/linux/man-pages/man1/top.1.html
[6] http://pubs.opengroup.org/onlinepubs/007904975/utilities/crontab.html
[7] https://tools.ietf.org/html/rfc4180

pripravíme šablóny pomocou nástroja *Googe Sheets*[8], v ktorých bude možné spracované dáta rezprezentovať pomocou grafov.

Nástroj *Google Sheets* sme si vybrali z dôvodu online prístupnosťi dokumentov, nezávislosti na operačnom systéme alebo používanom software, možnosti spolupráce viacerých užívateľov na jednom dokumente zároveň ale taktiež aj vďaka tomu, že tento nástroj je voľné dostupný bez nutnosti zakúpenia licencie a podporuje prevod do formátu používanym inými nástrojmi na reprezentáciu súborov vo frmáte *CSV*.

[8] https://www.google.com/sheets/about/

Kapitola 4

Implementácia

V tejto kapitole si predstavíme implementáciu vlastného systému na správu serverových fariem s použitím nástroja *Ansible*. Priblížime si problematiku implementácie jednotlivých modulov, ktoré sme si predstavili v kapitole 3.3 a nakoniec si ukážeme spôsoby testovania systému a ukážky testov.

4.1 Príprava systému

Tento nám uľahčuje a automatizuje prípravu systému pred začatím jeho používania. Skladá sa z dvoch komponentov, a to je:

- Vytvorenie inventárov
- Nasadenie SSH kľúčov

4.1.1 Vytvorenie inventárov

Pri vytvorení inventárov je potrebné spracovať dáta, v našom prípade vo formáte *CSV*, do formátu *Inventory* súborov s ktorými *Ansible* pracuje pri správe uzlov, a taktiež zaistiť ochranu citlivých údajov s ktorými pracujeme.

Povinnými údajmi na spracovanie sú `host`(reprezentuje názov alebo IP adresu uzlu) a `user` (užívateľské meno potrebné na prihlasovanie). Nepovinnými údajmi sú `password` (heslo - v prípade používania SSH kľúčov môže ostať nevyplnené) a `group`(predstavuje skupinu uzlov - v prípade, že je nevyplnené uzol nie je zaradený do žiadnej skupiny). Pri použití statického inventára je možné definovať ďalšie citlivé informácie, ktoré môžu byť použité ako premenné pri spúšťaní scenárov.

V nasledujúcich odstavcoch si popíšeme implementáciu skriptov pre jednotlivé druhy inventárov a metódy ochrany citlivých údajov pri danných druhoch inventárov. Pri implementácií obidvoch skriptov sme využili programovací jazyk *Python*.

Súbory vytvorené jednotlivými skriptami následne stačí skopírovať(respektíve pri spúšťaní skriptu zadať cestu kam sa majú uložiť) do adresáru `inventory` modulu, ktorý chceme následne používať.

```
host,user,password,group
test1.example.com,devops,sostrongpasswd,production
test2.example.com,user1,weakpasswd,production
```

Výpis kódu 4.1: Príklad vstupného súboru inventára vo formáte CSV

Statický inventár

Pri vytváraní statického inventára sme pripravili skript `generate_inventory.py`, ktorý prevedie dáta vo formáte *CSV* do formátu statických súborov.

Skript je možné spustiť v dvoch režimoch:

- **Nezabezpečený režim** - vytvorí sa len súbor `hosts`, ktorý zahŕňa názvy uzlov a skupiny do ktorých uzly patria.
- **Zabezpečený režim** - vytvorí sa súbor `hosts` a taktiež sa vytvorí adresár `host_vars` s podadresarmi obsahujúcim dáta pre jednotlivé uzly, ktoré su zašifrované. Na použitie šifrovania je potrebné spustiť skript s argumentom `--vault` ktorý vyzve užívateľa k zadaniu prístupového hesla a zabezpečí zašifrovanie údajov pomocou nástroja *Ansible Vault*. V rámci bezpečnosti a použitia metódy *DLP* o ktorej sme hovorili v kapitole 3.4 sme umožnili zadať heslo priamo do terminálu, ale aj formou cesty k súboru, ktorý obsahuje heslo. Ďalej navrhujeme v rámci dodržania maximálnej bezpečnosti, uchovávať heslo na odnímateľnom hardwarovom zariadení ako napríklad USB kľúči. Takisto doporučujeme aj nastaviť právo pristupovať k súboru obsahujúcom heslo len pre užívateľov s dostatočnými oprávneniami. V prípade použitia šifrovania je nutné spúšťať jednotlivé scenáre s parametrom `--ask-vault-pass`, ktorý vyzve k zadaniu šifrovacieho reťazca priamo alebo s parametrom `--vault-password-file` je možné definovať cestu k súboru s heslom.

```
$ANSIBLE_VAULT;1.1;AES256
383666656333386638366366393335636338623065323064313231366432323836336666
653430346664393636336462383332336532336335373634 0a373735343063643230373
636466616262613830353563653431363761616638333466336535396534623839363 73
316337643162323637 0a32343166373530613932653436623561343136623662333761 3
623133653630366135323262386662666132613363356466386539376566166132376 13
623965636432666363346130623933393964373733663133306 1
```

Výpis kódu 4.2: Príklad zašifrovaných dát pomocou nástroja Ansible Vault

Pri použití premenných, ktoré sme zašifrovali v scenároch, je nutné uviesť odkiaľ sa majú dáta pri použití scenáru alebo role čerpať, to znamená zmeniť zdroj dát z podadresára `defaults` alebo `vars` na šifrované súbory.

Dynamický inventár

Pri vytváraní dynamického inventáru sme pripravili skript `dynamic_inventory.py`, ktorý prevedie dáta vo formáte *CSV* do formátu *JSON* s ktorým za pomoci použitia argumentu `--list` vie *Ansible* pracovať.

Dynamický inventár funguje na princípe dynamického generovania inventory dát zo vstupných údajov. Tento spôsob vytvárania inventry súborov je oveľa rýchlejší a je omnoho jednoduhšie pozmeniť dáta avšak pri nižšej bezpečnosti, nakoľko citlivé dáta nie sú šifrované.

V rámci zaistenia lepšieho zabezpečenia dát odporúčame nespracované dáta uchovávať na oddelenom hardwarovom zariadení, a taktiež ich zašifrovať pomocou nástroja *Ansible Vault*. Bohužiaľ, *Ansible* momentálne neposkytuje možnosť použiť šifrované dáta pri používaní dynamického inventára a je potrebné ich manuálne odšifrovať pred použitím. V nasledujúcich verziách *Ansible* by mala byť aj táto funkcionalita podporovaná.

```
{
"group1": { hosts: [host1, host2] },
"_meta": {
       "hostvars": {
              "host1": {
                     "ansible_user" = user
                     "ansible_password" = password
              },
              "host2": {
                     "ansible_user" = user
                     "ansible_password" = password
              }
       }
}
```

Výpis kódu 4.3: Príklad výstupu dynamického inventára vo formáte JSON

4.1.2 Nasadenie SSH kľúčov

Ďalším úkonom pri nastavení systému je nasadzenie SSH kľúčov na spravované uzly. Vďaka tomu môžeme používať dynamický inventár bez uchovávania hesiel a jednotlivé uzly budú dostupné len zo zariadenia na správu, čím sa zvýši bezpečnosť systému. Pred prvým použitím je nutné aby inventory súbory obsahovali kompletné prístupové údaje k spravovaným uzlom tj. meno aj heslo.

Na nasadzovanie SSH kľúčov sme vytvorili scenár `setup_ssh_playbook.yaml` a rolu `ssh_key_deploy` pomocou ktorých sa automaticky nasadí SSH kľúč na uzly uvedé v adresári `inventory`. Scenár podporuje možnosť, kde sa najprv vytvorí alebo updatuje užívateľ pod ktorým má byť na zariadenie SSH kľúč nasadený a následne sa kľúč na spravovaný uzol nasadí. V podadresáry role môžeme definovať cestu k súboru obsahujúcemu verejný kľúč. V prípade, že verejný kľúč ešte nemáme vygenerovaný, je potrebné ho vygenerovať[1].

4.2 Nasadzovanie a konfigurácia software

Modul nasadzovania a konfigrácie softwaru sa delí na menšie logické celky ako sme si popísali pri návrhu na obrázku 3.2. Tieto logické celky sú reprezentované rolami, a tak je ich možné ľubovoľne kombinovať. Vytvorili sme taktiež viacero scenárov, kotré obsahujú najbežnejšie kombinácie jednotlivých roli ako napríklad nasadenie NodeJS servera s rôznymi typmi databázy a podobne. V prípade potreby iných kombinácií je ich možné jednoducho vytvoriť za pomoci pripravených roli. Pri spustení scenárov vidíme priebeh jednotlivých úloh na každom z uzlov a v prípade nastavenia aj debugovací výpis, ktorí nám poskytne detailnejšie informácie o priebehu. Nakonci scenára môžeme vidieť prehľadný súhrn úloh, ktoré sa podarilo previesť, respektíve informácie o chybách, ktoré nastali.

[1] https://www.ssh.com/ssh/key/

4.2.1 Webové služby

Medzi webové služby, ktoré sme implementovali patrí: *NodeJS* server a *Apache* server. Každý typ webovej služby je predstavovaný rolou, ktorá zabezpečuje inštalovanie potrebných programov a závislostí a následnú konfiguráciu v prípade potreby, napríklad povolenie firewallu a podobne. Pri každej role je možné špecifikovať nastavenia služby, respektíve rozšírenia, ktoré je k nej možné doinštalovať. V adresároch jednotlivých rolí sa nachádzajú podadresáre `vars` a `defaults`, ktoré obsahujú *YAML* súbory so základnou konfiguráciou a popisom možných nastavení.

Výsledkom použita rolí je funkčné prostredie pripravené na nasadenie aplikácií jednotlivých technológií.

4.2.2 Databázové služby

Medzi databázové služby, ktoré sme implementovali patrí: *MongoDB* a *MySQL* . Taktiež ako webové služby, každý typ databázy predstavuje samostatnú rolu. Role zabezpečuju nainštalovanie potrebného softwaru na beh databázových systémov, konfiguráciu, spustenie a vytvorenie databáz, respektíve vytvorenie užívateľov a prístupových údajov k databázam. Role obsahujú šablony pre konfiguračné súbory, ktoré je v prípade potreby možné upraviť buď manuálne alebo pomocou nastavenia premenných v podadresároch `vars` a `defaults`, ktoré obsahujú YAML súbory so základnou konfiguráciou a popisom možných nastavení. V týchto podadresároch je taktiež možné nastaviť rozšírenia, ktoré sa majú nainštalovať.

Výsledkom sú funkčné databázové systémy pripravené na použitie.

4.2.3 Správa systémových prvkov

Správa systémových prvkov obsahuje viacero druhov úkonov. Ide o úkony spojené priamo so systémom alebo systémovými aplikáciami. Tieto úkony su vykonávané pomocou systémovych modulov *Ansiblu*, ktoré nám umožňujú pracovať s jednotlivými druhmi úkonov. Pri všetkých rolách je možné upravovať ich nastavenia pomocou editovania premenných definovaných v súboroch v podadresároch `vars` a `defaults` jednotlivých rolí.

Tieto úkony zahŕňajú:

- **Správu užívateľov** - zahŕňa vytváranie, editovanie a mazanie užívateľov na základe nastavení role.
- **Správu skupín** - zahŕňa vytváranie, editovanie a mazanie skupín užívateľov na základe nastavení role.
- **Správu súborov** - zahŕňa sťahovanie a kompresiu alebo dekompresiu súborov podľa nastavení príslušnej role.
- **Správu úložiska** - zahŕňa nainštalovanie a konfiguráciu nástrojov *Docker*[2] a *Logical Volume Manager*[3], pomocou ktorých je možné rozširovať úložisko a vytvárať oddelené kontajnery systému. Role zahŕňajú kompletnú prípravu obidvoch nástrojov na použitie.
- **Zálohovanie** - zahŕňa vytvorenie zálohy súborov systému alebo inštalovaných databázových systémov.

[2] https://www.docker.com/
[3] https://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-lvm.html

- **Získanie zoznamu inštalovaných aplikácií** - zahŕňa získanie a stiahnutie zoznamu nainštalovaného software na spravovanom uzle.
- **Inštaláciu *EPEL* repozitárov** - zahŕňa inštaláciu *EPEL* repozitárov v závislosti na operačnom systéme.

4.2.4 Správa clustrov

Správa clustrov predstavuje úkony spojené so správou a vytváraním clustrov z jednotlivých serverov. Tak ako aj v predchádzajúcich prípadoch sú jednotlivé úkony implementované formou rolí. V rámci každej role môžeme špecifikovať premené, pomocou ktorých vieme upravovať defaultné nastavenia roli.

Medzi tieto úkony zaraďujeme inštaláciu a konfiguráciu nasledujúcich prvkov:

- **Vyrovnávač záťaže** - pomáha rozložiť záťaž medzi viaceré uzly clusteru. V našom prípade sme implementovali rolu, ktorá nainštaluje a nakonfiguruje *HAProxy*[4] vyrovnávač záťaže pre *HTTP* servery podľa zadaných nastavení umiestnených v podadresároch `defaults`.
- ***GlusterFS***[5] - je nástroj, ktorý slúži na pripravenie zdieľaného súborového systému naprieč viacerými servermi, ktoré tvoria cluster. Rola, ktorú sme pripravili, najprv nakonfiguruje potrebné nastavenia firewallu a následne nainštaluje a nakonfiguruje nástroj *GlusterFS*. Rola taktiež umožňuje automatické nastavenie a uchytenie (ang. mount) súborového systému.

4.3 Registrácia hardware

Modul registrácie hardwarového vybavenia v sebe zahŕňa zbieranie, interpretovanie a sumarizáciu dát o hardwarovom vybavení serverov.

Voči návrhu sme pri impementácií rozšírili možnosti získavania informácií o hardwarových komponentoch. Na získavanie týchto informácií sme si pripravili dva druhy scenárov pomocou ktorých na spravovaných zariadeniach nainštalujeme software na získanie informácií o hardwarových komponentoch v prípade, že ho systém neobsahuje. Následne scenáre zabezpečia prenos získaných dát na server použitý na správu. Pri každom scenári je možné definovať kam sa informácie na serveri na správu majú uložiť.

Prvý scenár s názvom `register_node_hw_via_ansible_facts.yaml` využíva na získanie informácií o spravovanom zariadení vstavanú funkcionalitu nástroja *Ansible* a to *Facts Gathering*. Pomocou tejto funkcionality získame dáta o hardwarovom vybavení uzlov vo formáte JSON, ktoré automaticky ukladame na zariadení na správu. Voľbu ukladania je možné nastaviť v súbore `ansible.cfg`, a to zmenením premennej `fact_caching_connection`. V prípade že premennú necháme zakomentovanú, údaje sa nebudú ukladať.

Druhý scenár názvom `register_node_hw_via_lshw.yaml` využíva pri získavaní dát nástroj *lshw*[6] pomocou ktorého získava dáta o hardwarovom vybaveí uzlu. Scenár najprv zistí dostupnosť nástroja *lshw* a v prípade, že nástroj nie je nainštalovaný, tak ho nainštaluje.

[4] http://www.haproxy.org/
[5] https://www.gluster.org/
[6] https://github.com/lyonel/lshw

Následne sú dáta o hardware vo formáte JSON stiahnuté zo spravovaného uzlu na server na správu, kde sú uložené pod názvom obsahujúcim informácie o názve uzlu a čase vykonania registrácie.

```
139.59.142.93__2018-05-02-22:02__hardware_list.json
46.59.142.93__2018-05-02-22:02__hardware_list.json
hp-moonshot-01.example.com__2018-05-02-18:44__hardware_list
```

Výpis kódu 4.4: Príklad vygenerovaných názvov súborov pri registrácií hardware

Následne sme pripravili skript `proces_hardware_informations.py`, ktorý prevedie vybrané dáta vo formáte JSON do formátu *CSV* a spojí dáta z viacerých uzlov do jedného kompletného súboru. Pri použití skriptu je nutné zadať typ zdroju dát, a to buď využitie nástroja *lshw* alebo *Ansible Gathering facts* a tiež aj zoznam súborov uzlov, ktoré chceme spracovať. Taktiež je možné pomocou argumentu skriptu `--type` špecifikovať o aké údaje máme záujem, ako napríklad informácie o CPU, pamäti RAM atď.

```
{ "id" : "hp-moonshot-01.example.com",
  "handle" : "DMI:0100",
  "product" : "ProLiant m360 Server Cartridge (734619-B21)",
  "vendor" : "HP",
  "serial" : "CN8352AFDASDU",
  "children" : [
    {
      "id" : "core",
      "children" : [
        {
          "id" : "cpu",
          "class" : "processor",
          "description" : "CPU",
          "product" : "Intel(R) Atom(TM) CPU C2750 @ 2.40GHz",
          "vendor" : "Intel Corp.",
          "width" : 64,
          "configuration" : {
            "cores" : "8",
            "threads" : "8"
          },
```

Výpis kódu 4.5: Príklad získaných dát o hardwarových komponentách

Skript vygeneruje *CSV* súbor, ktorý obsahuje dáta o zvolenom type komponentov, ktoré je možné následne importovať do nástrojov na prácu s tabuľkovými dátami ako napríklad *Google sheets* alebo *MS Excel*.

```
Host ID,CPU type,CPU Vendor,CPU Frequencies, ...
centos2,Intel(R) Xeon(R) CPU E5520 2.26GHz,Intel Corp., ...
hp-moonshot-01.example.com,HP,Intel(R) E5520 2.26GHz, ...
hp-moonshot-02.example.com,HP,Intel(R) E5520 2.26GHz, ...
hp-moonshot-02.example.com,HP,Intel(R) E5520 2.26GHz, ...
```

Výpis kódu 4.6: Príklad spracovaných dát s argumentom –cpu

K prehľadnému zobrazeniu dát formou grafov sme vytvorili šablonu[7], pomocou nástroja *Google sheets*, kam je možné importovať získané dáta a ich následné zobrazenie pomocou grafov.

4.4 Monitorovanie

V rámci monitorovania získavame údaje o stave softwarového a hardwarového vybavenia, ktoré následne ukladáme a poskytujeme štatistiku o behu jednotlivých uzlov. Na nastavenie monitorovania sme pripravili skript `manage_monitoring.py`, ktorý pripraví *Cron Job* pomocou ktorého sa bude automaticky spúšťať monitorovanie. Pri zadaní argumentu `--setup_hardware_monitoring` alebo `--setup_software_monitoring` je užívateľ vyzvaný k zadaniu parametrov o čase spúšťania monitorovania. Následne sú v nastavených intervaloch spustené scenáre, ktoré získavajú dáta o stave hardwaru a softwaru spravovaných uzlov. Bližší popis jednotlivých druhov monitorovania si popíšeme v nasledujúcich podkapitolách.

Pomocou skriptu je taktiež možné nastaviť automatické mazanie starých záznamov o monitorovaní pomocou argumentu`--setup_cleaning`. Nastavením tohoto parametru sa vytvorí ďalší *Cron Job*, ktorý zabezpečí mazanie starých súborov. Pri mazaní súborov je možné zadať adresár v ktorom sa mazanie bude vykonávať, časový interval v akom sa má mazanie opakovať a taktiež aj maximálny vek súborov ktoré majú byť vymazané. Mazanie sa vykonáva pomocou skrytých argumentov skriptu `manage_monitoring.py`. Mazanie je možné spustiť aj manuálne, a to pomocou argumentu `--cleaning_folder`. Ktorý vyhľadá v zadanom adresári súbory staršie ako je zadaným maximálny vek a vymaže ich.

4.4.1 Software

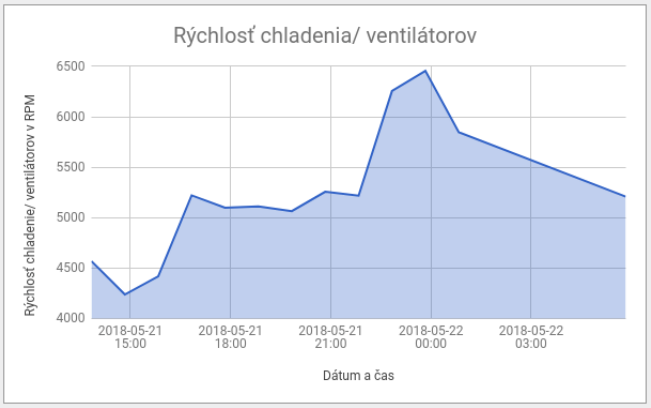
Na monitorovanie softwarového vybavenia slúži scenár `software_monitor_playbook.yaml`, ktorý je spúšťaný v pravidelných intervaloch na uzloch definovaných v inventory súbore. Pri monitorovaní softwaru sa kontroluje dostupnosť uzlov a či služby, ktoré sú špecifikované v podadresári `vars`, sú nainštalované a bežia na dannom uzle. V prípade, že sa zistí, že nejaká zo služieb nie je dostupná alebo nie je dostupný celý uzol, tak je vygenerovaný email ktorý je zaslaný na emailovú adresu správcu daného clustru alebo serveru. V podadresári `defaults` role *software_monitoring* je možné špecifikovať informácie o emailovej schránke odosielateľa a taktiež príjemcu. Správy je možné zasielať buď lokálne alebo s využitím emailového servera.

4.4.2 Hardware

Na monitorovanie hardwaroveho vybavenia slúži scenár `hardware_monitor_playbook.yaml`, ktorý získava zo spravovaných uzlov konkrétne dáta o stave hardwaru. Ako sme spomínali v kapitole 3.3 na základe analýzy sme si definovali aké dáta je potrebné získavať. Tieto dáta zahŕňajú informácie o využití porcesorov, teplote procesorov, stave a využití operačnej pamäte RAM, stave a využití diskov, či rýchlosti chladiacich jednotiek. Na získanie týchto informácií sme pripravili scenár, ktorý nainštaluje softwér *inxi*, potrebný na získavanie informácií o systéme. Následne sú za pomoci šablony upravené dáta do formátu *CSV* a potom sú stiahnuté na systém z ktorého prebieha správa.

[7] https://docs.google.com/spreadsheets/d/1np9Jor4qhIu-eYWEDgGxx4V18cEtIDOnl4K3fJ-RAbI/edit?usp=sharing

Nakoniec je možné pomocou argumentu `--merge_monitoring_data` | dáta z tých istých uzlov spojiť do jedného súboru a následne ich aplikovať na šablónu[8] pripravenú pomocou nástroja *Google Sheets* v ktorej uvidíme prehľad o výkone zariadenia včase. Pri spájaní súborov vznikne adresár a súbor pomenovaný podľa názvu jednotlivých uzlov.



Obr. 4.1: Výsledny graf zobrazujúci prvky monitorovania systému

4.5 Testovanie

V tejto podkapitole si popíšeme akým spôsobom sme systém testovali, a taktiež si ukážeme aj názorné ukážky práce so systémom.

Systém bol spúšťaný zo zariadení s operačným systémom *Fedora 27* a *CentOS 7.2*. Ako zariadenia na správu bolo použitých 14 serverov s operačným systémom *RHEL*, medzi ktoré patria servery :

- *HP Proliant DL360 G6* (2x Intel Xeon E5520 2.26GHz, RAM 32GB, SSD 2x146GB)
- *HP Proliant DL360 G7* (2x Intel Xeon X5650 2.66GHz, RAM 32GB, HDD 4x146GB)
- *DELL PowerEdge R330* (1x Intel Xeon E3-1230, RAM 16GB, HDD 4x1TB)

Testovanie prebiehalo priebežne v dvoch fázach, a to po naimplementovaní jednotlivých modulov systému a taktiež po naimplementovaní ich dielčich častí. Pri testovaní sme sa zamerali hlavne na nájdenie funkcionálnych chýb a zlepšenie funkcionality jednotlivých modulov.

Pri testovaní sme priebežne používali jednotlivé už implementované a otestované moduly na uľahčenie konfigurácie a zjednodušenie prípravy testovacieho prostredia. Napríklad po dokončení modulu na prípravu prostredia sme ho používali aj pri testovaní modulu na nasadzovanie systému. Vďaka tomuto prístupu sme odhalili niekoľko ďalších chýb v už kompletných moduloch a boli sme schopný ich opraviť.

Ukážky testovania a funkcie systému sa nachádzajú v prílohe B.

[8] https://drive.google.com/drive/folders/11owT4kSL3pDO23QzOAOr1DCBqJO2u5Xe?usp=sharing

Kapitola 5

Porovnanie s inými nástrojmi

V tejto kapitole si predstavíme najbežnejšie používané nástroje na správu serverov a serverových fariem a ich hlavné výhody a nevýhody. Následne sa pozrieme na porovnanie nami implementovaného riešenia a existujúcich nástrojov.

5.1 Puppet

Puppet, je nástroj typu klient/server, navrhnutý pre centralizovanú správu, konfiguráciu a údržbu operačných systémov a ich aplikácií. Poskytuje textové aj grafické používateľské rozhranie. Zaujímavosťou je, že aj systém , na ktorom beží serverová časť (*master*), môže vystupovať ako klientska časť (*agent*) ktorý je spravovaný inými strojmi[18].

Princíp fungovania *Puppetu* je, že sa stanoví výsledný stav, nie proces, akým ho dosiahneme. Tento proces pozostáva z vytvorenia súboru v jazyku *Puppetu*, ktorý sa nazýva *manifest*. Manifest popisuje stav serveru v akom by sa mal nachádzať. Na jeho základe agent realizuje operácie potrebné, aby bol systém v príslušnom stave. Inštaluje a konfiguruje aplikácie alebo odoberá tie, čo tam nemajú byť. V prípade spustenia *manifestu* na servery, ktorý je v aktuálnom stave voči manifestu, žiadne zmeny nebudú vykonané.

Tento prístup je čiastočne nevýhodou lebo vyžaduje neustále kontrolovanie a komunikáciu medzi jednotlivými strojmi a pomerne rozsiahle zmeny pri zmene konfigurácie.

Puppet používa špecifický programovací jazyk na princípe *DSL*[1] odvodený od jazyka *Ruby*. To znamená, že užívateľ musí ovládať syntax jazyka *Ruby*.

Výhody:

- podporuje takmer všetký operačné systémy
- jednoduchá inštalácia
- najkomplexnejšie užívateľské rozhranie

Nevýhody:

- komplikovanjšie úlohy vyžadujú použitie *CLI* a znalosť jazyka *Ruby*
- komplikované vytváranie náročnejších úloh a neprehľadné radenie úloh
- prístup založený na modeloch poskytuje menej kontroly

[1] https://en.wikipedia.org/wiki/Domain-specific_language

5.2 Chef

Chef slúži na správu konfigurácií a nasadzovanie softwaru. Funguje na princípe klient/server so špeciálnou vlastnosťou, a to tým že server potrebuje na svoje fungovanie oddelenú pracovnú stanicu na ovládanie. Pracovná stanica zašle na server súbor príkazov(kuchársku knihu) a server ich rozposiela ďalej na spravované uzly, ktoré na server následne posielajú odpovede. Komunikácia medzi spravovaným zariadením a serverom prebieha každých 30 minút, pričom sa zisťuje stav spravovaného uzla a vykonávaju sa úkony nutné k zaisteniu cieleného stavu.

Konfiguračné súbory sa v terminológií *Chefa* nazývajú recepty a sú usporiadané v kuchárskych knihách, ktoré predstavujú funkčné celky.

Výhody:

- veľké množstvo modulov a konfiguračných receptov
- prístup založený na kóde - zabezpečuje väčšiu kontrolu a flexibilitu konfigurácií
- možnosť spravovať fyzické alebo virtuálne kontajnery vo verejnom aj privátnom nasadení

Nevýhody:

- najkomplikovanejša inštalácia a konfigurácia systému
- nutnosť hlbšieho chápania jazyka *Ruby* a procedurálneho programovania
- vyžaduje nasadenie klientskej aplikácie a konfiguráciu v preddefinovanom pláne

5.3 Saltstack

SaltStack je nástroj, ktorý môže fungovať na princípe klient/server alebo aj ako necentralizovaný systém, v ktorom klientske systémy pracujú samostatne[19]. Umožňuje vzdialené nasadzovanie softwaru, správu konfigurácie, orchestráciu a monitorovanie systémov. V terminológií *SaltStacku* sa server na správu volá *master* a spravované uzly *salt minions*.

Minioni prímajú príkazy z mastra a po ich vykonaní zasielajú správu o stave späť. Pri komunikácií používajú *AES*[2] šifrovanie.

SaltStack umožňuje používanie zásuvných modulov, ktoré je možné implementovať ako modul v jazyku *Python*.

Výhody:

- jednoduché používanie po nastavení systému
- vstupné a výstupné súbory sú v rovnakom formáte - *YAML*
- vysoká škálovateľnosť a ovládateľnosť v prípade využitia master/minion modelu (možnosť použitia viacerých master zariadení v jednom systéme)

[2] https://www.webcitation.org/68GTcKdoD?url=http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf

Nevýhody:

- komplikované počiatočné nastavenie a konfigurácia systému
- neposkytuje gafické rozhranie (v súčasnosti vo vývoji)
- malá podpora nelinuxových systémov

5.4 Foreman

Foreman je trocha odlišný typ systému oproti predchádzajúcim. Je to systém, ktorý využíva na správu serverov a serverových fariem viacero druhov nástrojov a vďaka tomu umožňuje najefektívnejšie využitie funkcionality jednotlivých nástrojov v jednom systéme. Uvádzame ho tu pre porovnanie a ako ukážku iného prístupu k správe serverových fariem. Nebude zahrnutý v celkovom porovnaní, nakoľko využíva služby viacerých nástrojov, a tým pádom ho nevieme objektívne porovnať s ostatnými nástrojmi.

Foreman je projekt založený na otvorenom kóde, ktorý pomáha systémovým administrátorom spravovať servery počas ich celého životného cyklu, umožňuje automatizáciu nasadzovania software, správu konfigurácie, orchestráciu a monitorovanie systémov[20]. Umožnuje taktiež aj inštalovanie a prípravu operačných systémov alebo správu virtuálnych kontajnerov. Je napísaný v programovacom jazyku *Ruby* s využitím frameworku *Ruby on Rails*. Poskytuje užívateľom intuitívne grafické rozhranie a tiež má úplne zdokumentované a otestované aplikačné rozhranie (*REST API*) a textové rozhranie pre ovládanie z príkazovej riadky (*CLI*)[21].

Združuje viacero projektov založených na otvorenom kóde a poskytuje riešenia pre nasadzovanie, konfiguráciu a správu serverov. Projekty ktoré používa sú:

- *Foreman* - nasadzovanie
- *Pulp* - správa obsahu
- *Candlepin* - správa predplatného
- *Puppet* - správa konfigurácií a monitorovanie
- *Ansible* - nasadzovanie a správa konfigurácií
- *ElasticSearch* - indexovanie databázy
- *AMQP* - komunikácia medzi komponentami

Vďaka kombinácií viacerých nástrojov a využitiu ich funkcionality predstavuje *Foreman* veľmi komplexný a užitočný nástroj na správu serverových fariem.

5.5 Výsledky porovnania

Pomocou nami implementovaného riešenia s využitím nástroja *Ansible* sa nám podarilo dosiahnúť podobnú funkcionalitu ako ponúkajú ďalšie zmienené nástroje na správu serverových fariem.

Pri porovnaní sa teda pozrieme na rozdiely, ktoré sú medzi jednotlivými systémami, ako náročnosť systému na pamäťové požiadavky, náročnosť nasadenia systému a podobne.

Tieto rozdiely si predstavíme v tabuľke 5.1 a následne si zosumarizujeme výhody a nevýhody našeho riešenia oproti bežne dostupným nástrojom.

V tabuľke porovnávame prvky, ktoré sú podľa prieskumu 3.1 dôležité pri výbere nástroja na správu serverových fariem. Každý z nástrojov sme vyskúšali a na jednoduchých úlohach porovnávali náročnosť ich používania.

Metrika	**Puppet**	**Chef**	**SaltStack**	**Náš systém - Ansible**
Náročnost natavenia	Ťažká	Ťažká	Ľahká	Ľahká
Konfiguračný jazyk	DSL(PuppetDSL)	DSL(Ruby)	YAML(Python)	YAML(Python)
Náročnosť na pamäťové požiadavky	Veľké	Veľké	Malé	Malé
Náročnosť na používanie	Ťažká	Ťažká	Ľahká	Ľahká
Nutnosť nainštalovať agentov	Áno	Áno + ovládaciu stanicu	Oboje	Nie
Vysoká dostupnosť	Áno	Áno	Áno	Áno

Tabuľka 5.1: Porovnanie

Systémy *Puppet* a *Chef* sú veľmi komplexné systémy, ktoré ponúkajú veľkú funkcionalitu, ale ich ovládanie a nastavenie je dosť náročné a vyžaduje znalosti programovania. Pri porovnaní sme zistili, že medzi najlepšie systémy na správu pri zohľadnení sledovaných vlastností patrí *Ansible* a *SaltStack*, čo dokazujú aj prieskumy v používaní[3].

Výhody nami implementovaného systému v porovnaní s ostatnými nástrojmi :

- rýchle a jednoduché nastavenie
- jednoduchá obsluha
- nevyžaduje inštaláciu softwaru na strane spravovaných zriadení

Nevýhody nami implementovaného systému v porovnaní s ostatnými nástrojmi :

- nepoužívanie agentov na strane spravovaných zriadení, čo neumožňuje pri monitorovaní dozvedieť sa o chybe hneď pri jej vzniku (tento nedostatok je možné vyriešiť častejším spúšťaním monitorovacieho modulu)
- neposkytuje grafické užívateľské rozhranie (okrem výsledných dát grafov pri registrácií hardware a monitorovaní)
- umožňuje len základné monitorovanie systému (na komplexné monitorovanie je možné pomocou *Ansiblu* nainštalovať špeciálny monitorovací software, ako napríklad *Nagios*)

[3] https://stackshare.io/stackups/ansible-vs-chef-vs-puppet

Celkovo je nástroj *Ansible* vhodný najmä na nasadzovanie a konfiguráciu softwarových komponentov. Jeho použitie pri monitorovaní je obmedzené nutnosťou využívania dalších nástrojov.

Kapitola 6

Záver

Hlavným cieľom tejto bakalárskej práce bol návrh systému na správu serverových fariem (nasadzovanie, registráciu hardwarových komponentov a monitorovanie softwarovej a hardwarovej funkcionality), jeho implementácia s využitím nástroja *Ansible* a jeho následne porovnanie s existujúcimi nástrojmi na správu.

V teoretickej časti sme si popísali technológie použité pri implementácií, prostredie, ktoré sme použili a taktiež aj teoretické poznatky o problematike správy serverových fariem.

V praktickej časti sme sa zaoberali analýzou požiadaviek systému a na základe získaných dát z prieskumu medzi správcami serverových fariem sme vytvorili návrh modulov systému. Následne sme sa venovali detailom implementácie jednotlivých častí systému a ich testovaniu.

V poslednej časti sme porovnali implementovaný systém s už existujúcimi riešeniami na správu serverových fariem a predstavili si hlavné výhody a nevýhody nášeho riešenia.

Výsledkom práce je fungujúci systém na správu serverových fariem, ktorý umožňuje správu nasadzovania a konfigurácie softwaru, monitorovanie softwarovej a hardwarovej funkcionality systému a registráciu hardwarových komponentov serverov.

Testovanie prezentovaného riešenia sme vykonávali na skupine 14 fyzických serverov využívajúcich operačný systém *RHEL* alebo *CentOS*, na ktorých sme vykonávali hromadné nasadzovanie a konfiguráciu aplikácií, ako napríklad webových serverov a databázových systémov, registráciu hardwarových komponentov systémov a monitorovanie hardwarovej a softwarovej funkcionality.

Medzi možné rozšírenia systému pri ďalšom vývoji navrhujeme rozšíriť funkcionalitu na ďalšie operačné systémy a taktiež implementáciu grafického rozhrania na uľahčenie ovládania systému.

Pri porovnaní sme zistili, že nami implementovaný systém umožňuje podobnú funkcionalitu ako existujúce riešenia a to pri menších systémových nárokoch a jednoduchšom nastavení ako ostatné nástroje.

Avšak pri porovnaní sme zistili, že medzi najvhodnejšie systémy na správu serverových fariem radíme systémy ako *Foreman*, ktoré využívajú a kombinujú funkcionalitu viacerých nástrojov. Vďaka kombinácií funkcionality je možné optimalizovať postup pri správe systému.

Literatúra

[1] Plášil, F.; Staudek, J.: *Operační systémy*. Praha: SNTL-Nakladatelství technické literatury, prvé vydanie, 1992, ISBN 80-03-00269-9.

[2] Kabachinski, J.: Back to Basics: Understanding Operating Systems and Network Operating Systems. *Biomedical Instrumentation & Technology*, ročník 44, č. 5, 09 2010: s. 405–8, [Online; navštíveno 22.03.2018].
URL https://search.proquest.com/docview/759607642?

[3] User Interface - Operating system study guide. c2009-2015, [Online; navštíveno 24.04.2018].
URL http://faculty.salina.k-state.edu/tim/ossg/Introduction/ui.html

[4] Klika, P.: Historie UNIXu a Linuxu. c1999, [Online; navštíveno 04.04.2018].
URL https://www.fi.muni.cz/usr/jkucera/pv109/xklika-history.html

[5] Delozier, E. P.: The GNU/Linux desktop: an open access primer for libraries. *OCLC Systems & Services*, ročník 25, č. 1, 2009: s. 35–42, online; navštíveno 26.03.2018.
URL https://search.proquest.com/docview/209781387?

[6] Foundation, L.: The Top 10 Developers and Companies Contributing to the Linux Kernel in 2015-2016. ©2018, [Online; navštíveno 22.02.2018].
URL https://www.linuxfoundation.org/blog/the-top-10-developers-and-companies-contributing-to-the-linux-kernel-in-2015-2016/

[7] NOVEMBER 2017. c1993-2018, [Online; navštíveno 14.03.2018].
URL https://www.top500.org/lists/2017/11/

[8] Noyes, K.: Five Reasons Linux Beats Windows for Servers. ©2018, [Online; navštíveno 12.01.2018].
URL https://www.pcworld.com/article/204423/why_linux_beats_windows_for_servers.html

[9] Bookman, C.: *Linux clustering*. Boston: New Riders Publishing, prvé vydanie, 2003, ISBN 1-57870-274-7.

[10] Cluster Basics. ©2007, [Online; navštíveno 27.01.2018].
URL https://www.centos.org/docs/5/html/Cluster_Suite_Overview/s1-clstr-basics-CSO.html

[11] Top 5 High Availability Dedicated Server Solutions. 2017, [Online; navštíveno 27.01.2018].
URL https://www.accuwebhosting.com/blog/top-5-high-availability-dedicated-server-solutions/

[12] Ansible Documentation. 2017, [Online; navštíveno 22.02.2018].
URL http://docs.ansible.com/ansible/latest/index.html

[13] Ansible Documentation. ©2018, [Online; navštíveno 23.02.2018].
URL http://docs.ansible.com/ansible/latest/intro_installation.html

[14] Introduction To Ad-Hoc Commands. c2018, [Online; navštíveno 19.02.2018].
URL http://docs.ansible.com/ansible/latest/intro_adhoc.html

[15] Madhurranjan, M.; Raithatha, R.: *Learning Ansible*. Birmingham: Packt Publishing, prvé vydanie, 2014, ISBN 978-1783550630.

[16] Ansible Tower. c2018, [Online; navštíveno 20.02.2018].
URL https://www.ansible.com/products/tower

[17] Understanding and Selecting a Data Loss Prevention Solution. [Online; navštíveno 09.04.2018].
URL https://securosis.com/assets/library/reports/DLP-Whitepaper.pdf

[18] How Puppet works. c 2018, [Online; navštíveno 09.05.2018].
URL https://puppet.com/products/how-puppet-works

[19] Introduction to salt. ©2018, [Online; navštíveno 08.05.2018].
URL https://docs.saltstack.com/en/latest/topics/

[20] What is Foreman? [Online; navštíveno 10.05.2018].
URL https://www.theforeman.org/introduction.html

[21] Zapletal, L.: Klíčem k úspěšné správě IT infrastruktury je automatizace. *IT Systems*, , č. 1-2, 2015: s. 49–51, [Online; navštíveno 10.05.2018].
URL https://www.systemonline.cz/sprava-it/foreman-automatizuje-spravu-it.htm?mobilelayout=false

Prílohy

Zoznam príloh

Príloha A

Obsah přiloženého CD

Adresárová štruktúra CD:

- `management_system` - Zdrojové kódy systému na správu serverových fariem
- `docs` - Text práce vo formáte pdf a zdrojové súbory pre LaTeX
- `user_manual`- Užívateľský manuál vo formáte pdf

Príloha B

Ukážky testovania aplikácie

```
PLAY [Test] ********************************************************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com]
ok: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com]

TASK [system/epel_enable : Check if EPEL repo is configured] *******************************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "stat": {"atime": 1526937407.536161
2, "attr_flags": "", "attributes": [], "block_size": 4096, "blocks": 8, "charset": "us-ascii", "checksum"
: "6acc53a98eddeaef23b9d47b641030212331b257", "ctime": 1526846381.7731688, "dev": 64768, "device_type": 0
, "executable": false, "exists": true, "gid": 0, "gr_name": "root", "inode": 201364870, "isblk": false, "
ischr": false, "isdir": false, "isfifo": false, "isgid": false, "islnk": false, "isreg": true, "issock":
false, "isuid": false, "mimetype": "text/plain", "mode": "0644", "mtime": 1506966248.0, "nlink": 1, "path
": "/etc/yum.repos.d/epel.repo", "pw_name": "root", "readable": true, "rgrp": true, "roth": true, "rusr":
 true, "size": 951, "uid": 0, "version": "18446744072137482521", "wgrp": false, "woth": false, "writeable
": true, "wusr": true, "xgrp": false, "xoth": false, "xusr": false}}
ok: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": false, "stat": {"exists": false}}

TASK [system/epel_enable : Install EPEL repo when not configured - RHEL] *******************************
skipping: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "skip_reason": "Conditional r
esult was False"}
skipping: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": false, "skip_reason": "Conditi
onal result was False"}

TASK [system/epel_enable : Install EPEL repo when not configured - CentOS] *****************************
skipping: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "skip_reason": "Conditional r
esult was False"}
skipping: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": false, "skip_reason": "Conditi
onal result was False"}

TASK [system/epel_enable : Import EPEL GPG keys] *******************************************************
skipping: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "skip_reason": "Conditional r
esult was False"}
changed: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": true}

TASK [system/epel_enable : Debug output EPEL installation result] **************************************
skipping: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"skipped_reason": "Verbosity threshold not met.
"}
skipping: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"skipped_reason": "Verbosity threshold no
t met."}

PLAY RECAP *********************************************************************************************
hp-moonshot-01-c20.ml3.eng.bos.redhat.com : ok=2    changed=0    unreachable=0    failed=0
intel-wildcatpass-05.khw.lab.eng.bos.redhat.com : ok=3    changed=1    unreachable=0    failed=0
```

Obr. B.1: Výstup role inštalácie EPEL repozitárov

```
---

- name: Delete old docker versions
  package:
```

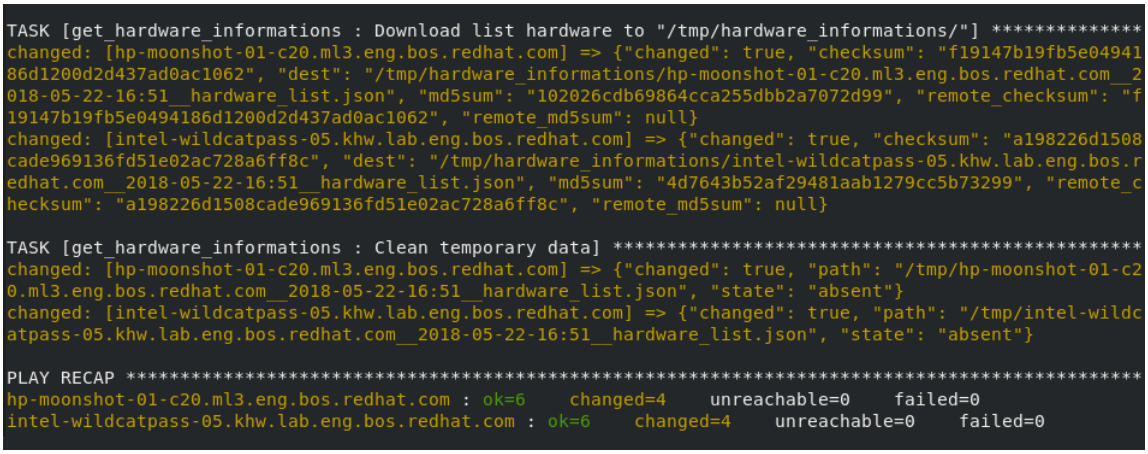
```
    name: '{{ item }}'
    state: absent
  with_items:
    - docker
    - docker-common
    - docker-engine

- name: Add docker GPG key
  rpm_key:
    key: https://download.docker.com/linux/centos/gpg
    state: present

- name: Download docker repo
  get_url:
    owner: root
    group: root
    mode: 0644
    url: https://download.docker.com/linux/centos/docker-ce.repo
    dest: /etc/yum.repos.d/docker-ce.repo

- name: Configure docker edge repo
  ini_file:
    dest: /etc/yum.repos.d/docker-ce.repo
    section: docker-ce-edge
    option: enabled
    value: '{{ docker_repo_enable_edge }}'

- name: Configure docker test
  ini_file:
    dest: /etc/yum.repos.d/docker-ce.repo
    section: docker-ce-test
    option: enabled
    value: '{{ docker_repo_enable_test }}'

- name: Install docker
  package:
    name: docker-ce
    state: present
  notify: restart docker

- name: Ensure docker is started
  service:
    name: docker
    state: started
    enabled: yes

- name: Add users to docker group
  user:
```

```
    name: "{{ item }}"
    group: docker
    append: yes
  with_items: "{{ docker_users }}"
```

Výpis kódu B.1: Rola inštalácie a konfigurácie nástroja Docker

```
PLAY [Test] ********************************************************************************

TASK [Gathering Facts] *********************************************************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com]
ok: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com]

TASK [system/storage/docker : Delete old docker versions] **********************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => (item=docker) => {"changed": false, "item": "docker",
"msg": "", "rc": 0, "results": ["docker is not installed"]}
ok: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => (item=docker) => {"changed": false, "item": "doc
ker", "msg": "", "rc": 0, "results": ["docker is not installed"]}
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => (item=docker-common) => {"changed": false, "item": "do
cker-common", "msg": "", "rc": 0, "results": ["docker-common is not installed"]}
ok: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => (item=docker-common) => {"changed": false, "item
": "docker-common", "msg": "", "rc": 0, "results": ["docker-common is not installed"]}
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => (item=docker-engine) => {"changed": false, "item": "do
cker-engine", "msg": "", "rc": 0, "results": ["docker-engine is not installed"]}
ok: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => (item=docker-engine) => {"changed": false, "item
": "docker-engine", "msg": "", "rc": 0, "results": ["docker-engine is not installed"]}

TASK [system/storage/docker : Add docker GPG key] ******************************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false}
changed: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": true}

TASK [system/storage/docker : Download docker repo] ****************************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "dest": "/etc/yum.repos.d/docker-ce
.repo", "gid": 0, "group": "root", "mode": "0644", "msg": "file already exists", "owner": "root", "secont
ext": "system_u:object_r:system_conf_t:s0", "size": 2428, "state": "file", "uid": 0, "url": "https://down
load.docker.com/linux/centos/docker-ce.repo"}
changed: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": true, "checksum_dest": null, "c
hecksum_src": "b9cab0cb1ddbffb77de51ba8838ea60335a229f6", "dest": "/etc/yum.repos.d/docker-ce.repo", "gid
": 0, "group": "root", "md5sum": "bbb0224eb355f307b39eed429c61be09", "mode": "0644", "msg": "OK (2424 byt
es)", "owner": "root", "secontext": "system_u:object_r:system_conf_t:s0", "size": 2424, "src": "/tmp/tmpO
3pFLZ", "state": "file", "status_code": 200, "uid": 0, "url": "https://download.docker.com/linux/centos/d
ocker-ce.repo"}

TASK [system/storage/docker : Configure docker edge repo] **********************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "gid": 0, "group": "root", "mode":
"0644", "msg": "OK", "owner": "root", "path": "/etc/yum.repos.d/docker-ce.repo", "secontext": "system_u:o
bject_r:system_conf_t:s0", "size": 2428, "state": "file", "uid": 0}
changed: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": true, "gid": 0, "group": "root"
, "mode": "0644", "msg": "option changed", "owner": "root", "path": "/etc/yum.repos.d/docker-ce.repo", "s
econtext": "system_u:object_r:system_conf_t:s0", "size": 2426, "state": "file", "uid": 0}

TASK [system/storage/docker : Configure docker test] ***************************************
ok: [hp-moonshot-01-c20.ml3.eng.bos.redhat.com] => {"changed": false, "gid": 0, "group": "root", "mode":
"0644", "msg": "OK", "owner": "root", "path": "/etc/yum.repos.d/docker-ce.repo", "secontext": "system_u:o
bject_r:system_conf_t:s0", "size": 2428, "state": "file", "uid": 0}
changed: [intel-wildcatpass-05.khw.lab.eng.bos.redhat.com] => {"changed": true, "gid": 0, "group": "root"
, "mode": "0644", "msg": "option changed", "owner": "root", "path": "/etc/yum.repos.d/docker-ce.repo", "s
```

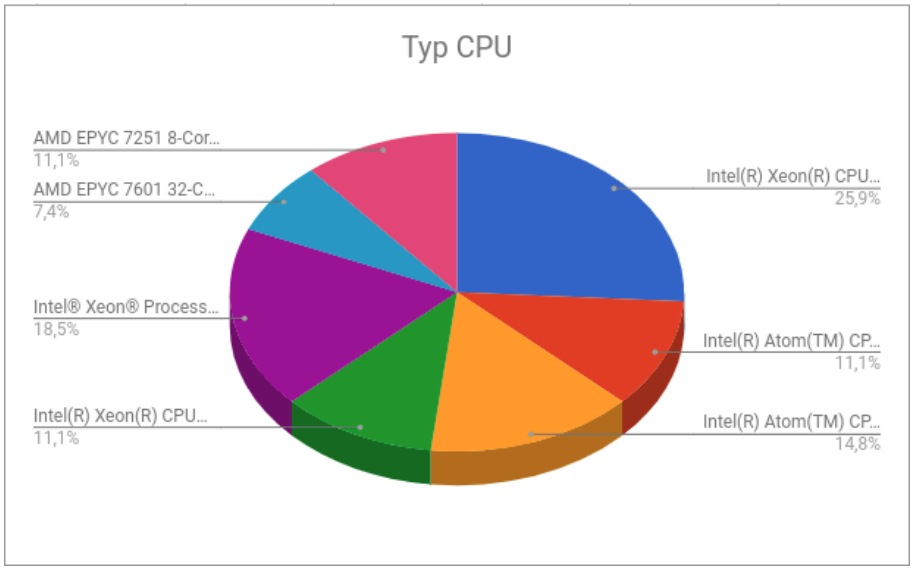
Obr. B.2: Výstup role inštalácie a konfigurácie nástroja Docker



Obr. B.3: Výstup role na registráciu hardwarových komponentov

Host ID	Vendor	System family	Architecture bits	Firmware	Number of CPUs	CPU type
server1.example.com	HP	ProLiant	64	BIOS	2	Intel Xeon X5650 2.66GHz
hp-moonshot-01.example.com	HP	ProLiant	64	BIOS	1	Intel Xeon E5520 2.26GHz
hp-moonshot-02.example.com	Dell	PowerEdge	64	BIOS	1	Intel Xeon E3-1230
hp-moonshot-03.example.com	HP	ProLiant	64	BIOS	1	Intel Xeon E5520 2.26GHz
server2.example.com	HP	ProLiant	64	BIOS	2	Intel Xeon X5650 2.66GHz
hp-moonshot-04.example.com	HP	ProLiant	64	BIOS	1	Intel Xeon E5520 2.26GHz
hp-moonshot-05.example.com	Dell	PowerEdge	64	BIOS	1	Intel Xeon E3-1230
server3.example.com	HP	ProLiant	64	BIOS	2	Intel Xeon X5650 2.66GHz
server4.example.com	HP	ProLiant	64	BIOS	2	Intel Xeon X5650 2.66GHz
hp-moonshot-06.example.com	Dell	PowerEdge	64	BIOS	1	Intel Xeon E3-1230
hp-moonshot-07.example.com	HP	ProLiant	64	BIOS	1	Intel Xeon E5520 2.26GHz
hp-moonshot-08.example.com	Dell	PowerEdge	64	BIOS	1	Intel Xeon E3-1230
server5.example.com	HP	ProLiant	64	BIOS	2	Intel Xeon X5650 2.66GHz

Obr. B.4: Dáta získane pomocou role na registráciu hardwarových komponentov



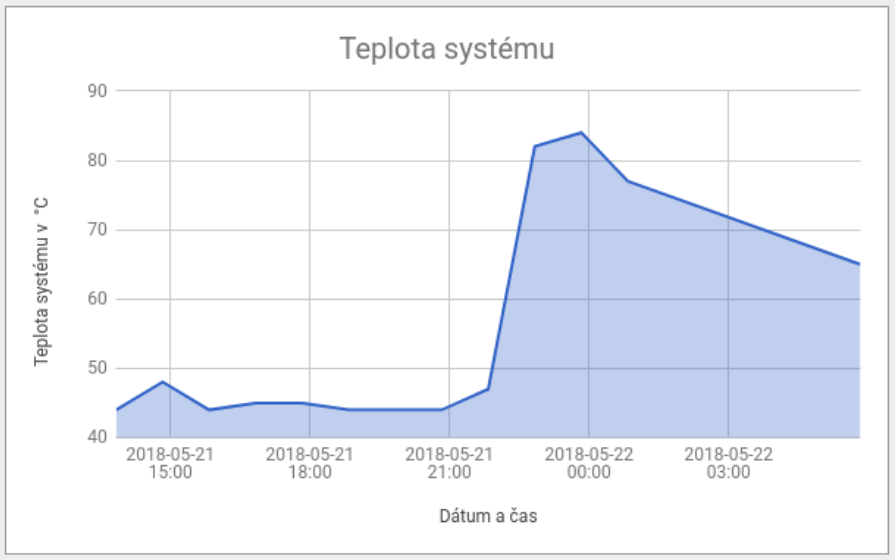
Obr. B.5: Dáta o registrácií hardware reprezentované formou grafu

```
–name: Get informations about system temperature
  shell: >
                inxi –s |
                grep 'System Temperatures'|
                grep –o –P '(?<=cpu).*'|
                grep –o –P '(?<= ).*(?= )'|
                grep –o –P '.*(?= )'|
                sed s'/.$//'

  register: system_temperature

– name: Get informations about fan speed
  shell: >
                inxi –s |
                grep 'Fan Speeds'|
                grep –o –P '(?<=cpu).*'|
                grep –o –P '(?<= ).*'|
                sed s'/.$//'
  register: fan_speed

– name: Get informations about system uptime
  shell: >
                inxi –I |
                grep 'Info'|
                grep –o –P '(?<=Uptime).*(?=Memory)'|
                grep –o –P '(?<= ).*(?= )'
  register: uptime
```

Výpis kódu B.2: Časť role modulu hardwarového monitorovania

Využitie uloziska (%)	Celková operačná pamäť (MB)	Využitá operačná pamäť (MB)	Využitie procesora (%)
7.00%	31883.4	602.5	20.9769
7.00%	31883.4	603.3	20.5371
7.00%	31883.4	603.6	20.4251
7.00%	31883.4	603.4	19.9445
7.00%	31883.4	802.5	19.8923
7.00%	31883.4	743.5	19.7804
7.00%	31883.4	703.8	19.6845
7.00%	31883.4	603.4	19.6501
7.00%	31883.4	601.8	18.2554
7.00%	31883.4	610.9	18.2583
7.00%	31883.4	853	18.255
7.00%	31883.4	1687.3	17.641
7.00%	31883.4	1007.6	15.5748

Obr. B.6: Dáta o monitorovaní hardwaru reprezentované v tabuľke



Obr. B.7: Dáta o monitorovaní hardwaru reprezentované formou grafu