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

AUTOMATICKÁ KONFIGURACE VIRTUÁLNÍ INFRASTRUKTURY
AUTOMATIC CONFIGURATION OF VIRTUAL INFRASTRUCTURE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Kadlíček**
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 **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Jan Kadlíček ***ID:*** 186524
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Automatická konfigurace virtuální infrastruktury

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte si problematiku automatizace infrastruktury, zaměřte se jak na počáteční instalaci a nastavení serveru/platformy, tak i na následné změny při běhu aplikací. Srovnejte dostupné nástroje (jako jsou například Ansible, Terraform, Puppet) a na základě požadavků od vedoucího vyberte nejvhodnější nástroj. S využitím nástroje pak vytvořte konfigurační soubory pro nasazení scénáře s několika službami, jejich monitorováním, klienty a sítěmi pro prostředí Docker. U výsledného scénáře změřte jeho náročnost na zdroje a výsledky vyhodnoťte. Implementujte řešení, které dokáže automaticky vytvořený scénář smazat.

DOPORUČENÁ LITERATURA:

[1] HOCHSTEIN, Lorin, Scott LOWE a Matt OSWALT. Ansible: up and running. Beijing: O'Reilly, [2015]. ISBN 14-919-1532-3.

[2] BRIKMAN, Yevgeniy, Scott LOWE a Matt OSWALT. Terraform: up and running : writing infrastructure as code. Sebastopol, CA: O'Reilly Media, 2017. ISBN 14-919-7708-6.

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

Tato diplomová práce se zaměřuje na problematiku automatizace virtuální infrastruktury a dostupné nástroje jako jsou například Ansible, Terraform či Puppet. Jednotlivé nástroje jsou zde posupně porovnávány v rámci instalací či vytváření konfigurací. Výsledkem této diplomové práce je pak výběr vhodného nástroje pro automatizaci virtuální infrastruktury.

Klíčová slova

Virtuální infrastruktura, operační systém, klient, server, konfigurace, konfigurační soubor, playbook, host, modul, Ansible, Terraform, Puppet, Nginx, WordPress, MySQL, FTP, Apache, Lighttpd

Abstract

This master thesis focuses on the issue of automation of virtual infrastructure and available tools such as Ansible, Terraform and Puppet. The individual tools are compared here in sequence for installations or configuration creation. The result of this master thesis is the selection of a suitable tool for automation of virtual infrastructure.

Keywords

Virtual infrastructure, operating system, client, server, configuration, configuration file, playbook, host, module, Ansible, Terraform, Puppet, Nginx, WordPress, MySQL, FTP, Apache, Lighttpd

Bibliografická citace:

KADLÍČEK, Jan. *Automatická konfigurace virtuální infrastruktury* [online]. Brno, 2020 [cit. 2020-05-24]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/126034. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Václav Uher.

Prohlášení

„Prohlašuji, že svou závěrečnou práci na téma Automatická konfigurace virtuální infrastruktury jsem vypracoval samostatně pod vedením vedoucího semestrální práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne **1. června 2020** ……

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Václavu Uhrovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne **1. června 2020** ……

podpis autora(-ky)

Obsah

Seznam obrázků

Seznam souborů

Seznam tabulek

Seznam grafů

ÚVOD

Tato semestrální práce se bude zabývat problematikou týkající se automatizace virtuální infrastruktury, konkrétně nástroji, které tento druh automatizace umožňují jako je Ansible, Terraform či Puppet. Jednotlivé kapitoly této práce se tak budou zabývat těmito nástroji. První kapitola však bude věnována technologii virtualizace, která se bude v rámci této práce využívat. Následující kapitoly pak budou věnovány společnostem poskytující tyto nástroje, dalším bodem této práce bude instalace a případná konfigurace těchto nástrojů. Pomocí nástrojů pro automatizaci infrastruktury pak budou následně vytvořené jednoduché konfigurace, pomocí kterých se vyzkouší funkčnost jednotlivých nástrojů. Na základě všech získaných poznatků a zkušeností dojde ke zmapování a následnému porovnání těchto tří dostupných nástrojů. Na základě těchto poznatků se vybere nejvhodnější nástroj, s jehož pomocí se vytvoří jednoduchá ukázková konfigurace serveru.

Informace potřebné pro tuto práci budou čerpány z různých dostupných zdrojů, jako jsou například knihy, články, dokumentace výrobců těchto produktů a další.

Hlavním přínosem práce je porovnání nástrojů pro automatizaci virtuální infrastruktury a následná realizace scénářů virtuální infrastruktury s pomocí nejvhodnějšího nástroje.

K testování bude využita virtualizace a virtualizační nástroj VirtualBox. Prostřednictvím tohoto nástroje budou vytvářeny virtuální stroje v podobě klienta a serveru. Dále bude v rámci tohoto nástroje vytvořena virtuální síť pouze pro hosty, která bude zajišťovat vzájemné propojení klienta a serveru. Pro klientský systém bude využit operační systém Ubuntu Desktop ve verzi 19.04 a 19.10, v případě serveru pak Ubuntu Server ve verzi 19.04 a 19.10.

1 VIRTUALIZACE

Virtualizaci je možné označit jako techniku, popřípadě způsob, jak na jednom fyzickém stroji spustit další stroj či stroje virtuální. Virtualizace nám poskytuje jakousi virtuální vrstvu, díky které je možné na jednom fyzickém hardware virtualizovat například software v podobě operačního systému, popřípadě virtualizovat jednotlivé hardwarové komponenty jako jsou například paměť, procesor a další. Tato vrstva tak umožňuje propojení (komunikaci) virtuálního stroje (např. virtualizovaný operační systém) a fyzického hardware. Virtualizace však není jen jakousi technikou pro spuštění nějakého virtuálního stroje, ale může také plnit i tzv. bezpečnostní funkci, kdy pomocí virtualizace je virtualizovaný stroj (například virtuální operační systém) oddělený a nezávislý na hostitelském operačním systému, případně hardwaru, pokud se jedná o virtualizaci hardwaru.

1.1 Historie

Z hlediska historie se nejedná o nějakou technologii, jelikož první nasazení virtualizace se datuje kolem 60. let minulého století americkou společností IBM. Tato společnost představila operační systém, který umožňoval vytvořit dílčí virtuální stroje. IBM tento software nasadila u svých sálových počítačů IBM 704, kdy tyto stroje navrhnuté pro vědecké účely neumožňovaly provádět současně několik výpočtů (operací). Z tohoto důvodu se tak společnost IBM rozhodla vyvinout a nasadit software, který umožní vytvořit několik jednoduchých virtuálních strojů a provádět tak několik výpočtu současně. [1], [2]

Nasazení virtualizace v osobních počítačích dlouho nebyla vůbec možná, protože hardware tehdejších osobních počítačů nebyl vůbec připravený a zároveň nebyl dostatečně výkonný. Vývoj současné virtualizace v rámci osobních počítačů odstartovala na konci devadesátých let kolem roku 1998 americká společnost VMware, která se dodnes zabývá prodejem a vývojem virtualizačních nástrojů pro osobní počítače, servery a cloudovými řešeními. O pár let přichází společnost Intel s novými procesory, které nově podporují virtualizaci (Intel VT-x), v zápětí přichází AMD s novými procesory podporující virtualizaci (AMD-V). Roku 2007 je vydán nový virtualizační nástroj VirtualBox patřící společnosti Oracle Corporation, jedná se o přímého konkurenta virtualizačních nástrojů společnosti VMware, konkrétně nástrojů VMware Workstation Pro/Player. V současné době přicházejí nejrůznější nové techniky a typy virtualizace jako je například kontejnerová virtualizace a další. Některé z těchto novějších či nejnovějších technik dokážou v některých případech nahradit i klasickou (úplnou) virtualizaci. Objevují se i nové nástroje pro správu či údržbu virtuálních strojů. Záleží tak dnes na uživateli, pro kterou technologii se

rozhodne, popřípadě která je pro něj nejvhodnější z hlediska jejího nasazení nebo podle samotných uživatelových zkušeností s danou technologií. [3], [4]

1.2 Základní virtualizační technologie

Jak již bylo zmíněno v předcházející kapitole, virtualizace celkově za dobu od jejího vzniku prošla vývojem, největším však za posledních asi dvacet let, kdy virtualizaci je možné použít i na osobních počítačích, a mohou tak vznikat nejrůznější nové technologie virtualizace. Tato kapitola bude pojednávat o základních technikách a typech virtualizace. V následujících kapitolách pak budou tyto jednotlivé druhy virtualizačních technik blíže popsány včetně jejich výhod a nevýhod, možností jejich využití, popřípadě jejich nasazení, a nakonec zde budou uvedeni výrobci, kteří jednotlivé nástroje poskytují.

1.2.1 Virtualizace aplikací

Jak již vyplývá z názvu, jedná se o technologii, která dokáže virtualizovat jednotlivý software například počítačové programy. Typickým příkladem virtualizace jsou tzv. portable (přenosné) aplikace či programy, které není nutné integrovat do systému, a proto jsou přenosné.

Pokud bychom chtěli takovouto přenosnou aplikaci vytvořit, je k tomu nutné použít příslušný software. Následně v tomto software postupujeme tak, že nejdříve zvolíme aplikaci, kterou chceme virtualizovat, poté je nutné vytvořit virtuální vrstvu, kdy software získá nastavení virtualizované aplikace, dynamické knihovny, konfigurační soubory a potřebné položky z registrů. Výslednou portable aplikaci je možné si představit jako adresář, popřípadě složku, ve které je umístěna nainstalována virtualizovaná aplikace a další soubory nutné k jejímu nezávislému spuštění. [5]

Výhodou tohoto typu virtualizace je zejména jeho přenositelnost ať už v rámci USB disku, tak například nějakého serveru. Nadále je možné využít tuto virtualizaci kvůli vzájemné nekompatibilitě aplikace a operačního systému, kdy starší verze aplikace nepodporuje novou verzi operačního systému. Další výhodou může být i bezpečnost, kdy můžeme nastavit, aby virtualizovaná aplikace nezasahovala do operačního systému. Nastavení je prováděno při vytváření virtuální aplikace, kdy je možné vybrat ze dvou režimů fungování virtuální aplikace. První možností je, že této aplikaci umožníme čtení z hostitelského systému a zároveň i zápis do tohoto systému, druhou možností je, že této virtuální aplikaci umožníme pouze čtení z hostitelského systému a není možné tak touto aplikací v hostitelském systému provádět nějaké změny. [5]

Mezi nástroje, které tento typ virtualizace umožňují patří například software Evalaze nebo ThinApp od společnosti VMware.

1.2.2 Virtualizace serverů

Jedná se klasickou technologii virtualizace například operačních systému Windows, Linux v platformách x64 či x86. Je to asi nejznámější virtualizační technologie, zejména u uživatelů osobních počítačů. Tento druh virtualizace je možné dále rozdělit na softwarovou a hardwarovou virtualizaci, tyto zmíněné druhy budou detailněji popsány v následující podkapitole.

Softwarová virtualizace

K realizaci softwarové virtualizace je potřebný hostitelský systém. V případě osobního počítače tak máme hardware, na kterém je nainstalován tento hostitelský systém. Nad operačním systémem běží virtuální vrstva, která zajišťuje komunikaci mezi virtuálním operačním systémem a hardware zmíněného osobního počítače. Tato metoda je tak vhodná pro odzkoušení nějakého software, Ať už nového operačního systému či nějakého programu. Nevýhodou může být fakt, že pro tento typ virtualizace je nutný hostitelský operační systém. Mezi nejznámější nástroje pro softwarovou virtualizaci patří VirtualBox, VMware Workstation Pro/Player, Qemu. [6]

Hardwarová virtualizace

Hardwarová virtualizace podobně jako softwarová virtualizace nabízí možnost virtualizovat operační systémy s platformami x64 nebo x86. Oproti softwarové virtualizaci odpadá nutnost využívat hostitelský operační systém. Princip této metody spočívá v tom, že na hardware (např. server) je rovnou instalován tzv. hypervisor, který zde zastává stejně jako u softwarové virtualizaci roli prostředníka, který zajišťuje komunikaci mezi fyzickým hardware a virtualizovaným operačním systémem. Velkou výhodu této technologie je menší náročnost na hardware a větší jednoduchost, což znamená menší počet chyb či aktualizací a je tak možné tuto technologii využít u serverů. Mezi výrobce, kteří tuto technologii nabízejí patří společnost VMware s nástrojem VMware vSphere. [6]

1.2.3 Virtualizace na úrovni operačního systému

Tento typ virtualizace nám umožňuje vytvoření virtuálních strojů, které jsou vzájemně od sebe odděleny. Tyto oddělené systémy se nazývají kontejnery. Virtualizaci na úrovni systému je tak možné nazývat i jako tzv. kontejnerovou virtualizaci. Jak už napovídá název, tento druh virtualizace je závislý na hostitelském operačním systému, využívá jádro operačního systému. Na hostitelský operační systém například nějaká distribuce systému Linux je nainstalována virtualizační vrstva, která umožňuje vytváření a spouštění kontejnerů. Kontejnerovou virtualizaci je možné realizovat i na operačních systémech Windows, v tomto

případě však za pomocí technologie Hyper-V. Výhodami této technologie je zejména rychlost, úspora místa na pevném disku a menší hardwarové nároky oproti například softwarové virtualizaci a virtualizovaným systémům. Velkou nevýhodou kontejnerové virtualizace je tak nutná závislost na hostitelském operačním systému, může docházet k pádům tohoto systému nebo případné nestabilitě v důsledku instalace nových aktualizací. [7]

Mezi nejznámější nástroje pro kontejnerovou virtualizaci patří Docker a Linux Containers (LXC). První verze Dockeru byla uvolněna roku 2013 jako open-source, kdy tento nástroj využíval pro své prostředí LXC, poději ale Docker navrhl nový ovladač, který je napsán v programovacím jazyku GO. [8]

1.2.4 Virtualizace uložišť

Tento druh virtualizace souvisí zejména s technologií RAID. První zmínka o této technologii se datuje na rok 1988, za tuto dobu prošla značným vývojem a dnes tak existuje několik verzí této technologie. Technologie diskových polí RAID je spojena zejména se servery, je možné říct, že se jedná o jakousi správu či řízení fyzických disků, které jsou umístěny v daném serveru. Několik těchto uložených fyzických disků se navenek chovají jako jeden velký disk. Pomocí technologie RAID je možné dosáhnou například vyšších přenosových rychlostí, zamezit ztrátám dat. V případě technologie RAID 0 jsou data při kopírování rozdělena a každá část je nakopírována na jeden disk, tím je umožněno naplno využít přenosových rychlostí všech disků. V případě zabezpečení proti ztrátě dat je nutné použít alespoň dva disky a technologii RAID 1, kdy jsou data uložena na obou discích a v případě ztráty jednoho z disků jsou data stále uložena na druhém disku (zrcadlení). V případě RAID 5 je potřeba použít alespoň tři disky, na které se ukládají paritní bity, v případě výpadku jednoho disku je možné data zachránit. [9], [10]

1.2.5 Virtualizace sítí

Obecně lze virtualizaci sítí definovat jako soubor hardwarových a softwarových prostředků, které jako celek tvoří virtuální síť. Vytvoření virtuální sítě VLAN spočívá rozdělení jedné sítě na tzv. dílčí podsítě nejčastěji v rámci přepínače. Díky tomu je možné nastavit provoz pouze v rámci jedné podsítě, popřípadě je se dá použít i v případě zabezpečení. Virtuální sítě je možné vytvářet i ve virtualizačních nástrojích pro softwarovou virtualizaci jako jsou Virtualbox či VMware Workstation PRO/Player. V rámci těchto nástrojů si uživatel může vytvořit různé virtuální sítě, kdy virtuální stroj je možné připojit zároveň k těmto několika sítím. [11]

2 NÁSTROJE PRO AUTOMATIZOVANÉ ŘÍZENÍ VIRTUÁLNÍ INFRASTRUKTURY

Tato kapitola bude nejdříve zaměřena na infrastrukturu jako takovou, bude zde postupně rozebrána definice virtuální infrastruktury, její zaměření, využití, výhody a nevýhody. Následně se tato kapitola bude zabývat možnostmi automatizace, to znamená, jaké jsou možnosti automatizace infrastruktury, kteří výrobci se těmito technologiemi zabývají a jaký software poskytují. Další podkapitoly tak budou zaměřeny na jednotlivé nástroje umožňující automatizovanou konfiguraci virtuální infrastruktury jako jsou například Ansible či Terraform. Po seznámení s jednotlivými nástroji budou tyto nástroje porovnány na základě jejich kladů a záporů, možnostech jejich nasazení a využití bude vybrán ten nejvhodnější pro další nasazení v rámci této závěrečné páce.

Infrastruktura se dá jednoduše popsat jako složení všech možných a dostupných prvků v rámci daného prostředí, což může být například síť s hardwarovými prvky (servery, počítače), síťovými prvky (routery, switche), službami a dalšími prvky. Jednotlivé infrastruktury se mohou lišit v rámci firem, organizací, institucí a jejich jednotlivých požadavků na danou síť. V případě jednotlivých organizací či firem by se měli u infrastruktur brát v potaz věci spojené se správnou volbou hardware, virtualizačních technologií a celkovému návrhu dané infrastruktury. Je nutné brát v potaz i volbu vhodného software ať už jednotlivého softwaru (např. programů), tak i operačních systémů. V rámci dané infrastruktury by pak měla být zajištěna i její správa, kdy je vhodné nasadit co nejefektivnější řešení, kdy je nutné nainstalovat například několik stejných operačních systémů. Jestliže je těchto zařízení několik, je možné tyto zařízení instalovat a konfigurovat jednotlivě, ale v případě velké organizace by toto řešení bylo velmi neefektivní a téměř nemožné. Tento problém se dá řešit například vytvořením jedné konfigurace systému, kde následně se odstraní všechny nepoužívané možnosti a funkce a pak teprve tento tzv. ořezaný systém nainstalovat. Mnoho firem však v budoucnu začne naplno využívat cloudových prostředí a bude stávající infrastrukturu značně rozšiřovat. V těchto případech může pomoci nasazení technologií pro automatizovanou správu virtuálních nebo i fyzických infrastruktur. Tyto technologie fungují tak, že správce vytvoří konfigurační soubor, prostřednictvím kterého je možné na dálku vytvořit několik virtuálních počítačů či serverů, dále je možné vzdáleně spravovat provoz již vytvořených strojů ať už jednotlivě, všech zároveň nebo vytvořit skupinu hostů, které se budou vzdáleně konfigurovat. Vzdálený přístup může být realizován například přes lokální síť, prostřednictvím SSH. Následující kapitoly budou dále věnovány již zmíněným nástrojům pro vzdálenou správu virtuální infrastruktury. [12], [13], [14]

2.1 Ansible

Ansible je nástrojem americké společnosti Red Hat sídlící ve městě Raleigh v Severní Karolíně. Zabývá se především produkcí linuxových distribucí jako jsou například Red Hat Enterprise Linux, Red Hat Enterprise Virtualization a další, zabývá se i poskytováním cloudových služeb nebo služeb spojenými s konzultacemi, certifikacemi, popřípadě zaškolováním. Společnost Rad Hat založili Bob Young a Marc Ewing v srpnu roku 1999 spojením dvou menších firem, které vlastnili tito dva členové. Roku 1993 Bob Young zakládá ACC Corporation, která zajišťovala doplňky pro operační systém Linux a Unix. Roku 1994 Marc Ewing vyvíjí vlastní linuxovou distribuci Red Hat Linux. Roku 1995 Bob Young odkupuje Ewingův software a společně zakládají společnost Red Hat. V roce 1999 Red Hat odkupuje společnost Cygnus Solutions, která zajišťovala programátory produktů GNU, podporu pro svobodné verze software či různý ladící software jako je například debugger. V červnu roku 2006 přechází pod Red Hat společnost JBoss, která se specializovala na vývoj middleware aplikací. Od roku 2008 spadá pod Red Hat i společnost Amentra poskytující systémové služby. V případě Red Hat se tak jedná o spojení několika menších firem, které se zabývaly vývojem jak linuxových, tak svobodných aplikací či systémů nebo zajišťováním jejich podpory. V červenci roku 2019 oznámily společnosti Red Hat a IBM dohodu o spolupráci, IBM odkoupila akcie Red Hat zhruba za 34 milionů dolarů. [15], [16]

Od roku 2004 americká společnost Red Hat působí i v České republice a roku 2006 oficiálně otevírá pobočku v Brně. Dnes má brněnská pobočka více než 700 zaměstnanců, jedná se tak o největší pobočku Red Hat v případě hlavního bydliště společnosti mluvíme o více jak sedmi tisících zaměstnancích. [17]

Pokud však budeme mluvit o samotném nástroji Ansible, jedná se o svobodný software, který vytváří platformu pro správu a řízení vzdálených počítačů. První verze byla uvedena v únoru 2012 firmou AnsibleWorks, kterou později roku 2015 odkoupila společnost Red Hat. Jak již bylo zmíněno Ansible umožňuje vzdáleně prostřednictvím spojení SSH či přes PowerShell provádět správu nebo konfiguraci. Je tak možné například na dálku nainstalovat a nakonfigurovat server. Mezi výhody nástroje Ansible patří zejména minimální nároky na potřebné místo v rámci instalace a na minimální potřebný software, pro spojení mezi hlavním a vzdáleným strojem je využíván SSH protokol, ke konfiguraci jsou využívání soubory psané v jazyce YAML, přesněji se jedná o formát pro serializaci dat textových souborů, který musí mít určitou strukturu a v případě špatného zápisu není vzdálená konfigurace, popřípadě správa spuštěna. Výsledná konfigurace vytvořená nástrojem Ansible je složena ze tří základních částí, a to z modulů, seznamu hostů a konfiguračního souboru (playbook). Jednotlivé části budou detailně popsány v rámci následující kapitoly. Díky modulům má tento nástroj různou škálu jeho

využití. Ansible je možné provozovat na operačních systémech Linux či macOS. V případě, že by uživatel chtěl Ansible instalovat na operační systém Windows, může v rámci distribuce Windows 10 využít technologii WSL (Windows Subsystem for Linux) a prostřednictvím této technologie nainstalovat například operační systém Ubuntu a vněm následně pak samotný Ansible. Nástroj umožňuje vytvářet konfigurace pro systémy Linux jako je například Ubuntu, CentOS, openSUSE a další distribuce, Microsoft v podobě Windows či platformy Microsoft Azure nebo macOS. Poskytuje tak nejrůznější moduly pro tyto operační systémy. I když Ansible poskytuje velké množství modulů, je možné vytvářet i moduly vlastní, uživatel si tak může vytvářet své vlastní postupy pro jednotlivé konfigurace v případě, že mu poskytnuté moduly nevyhovují nebo nesplňují jeho požadavky. Aby bylo možné spouštět vlastní moduly je nutné doinstalovat rozšiřující balíčky programovacího jazyka Python a zároveň klientský systém nakonfigurovat, aby Ansible využíval nově vytvořené moduly. Vlastní modul se vytvoří prostřednictvím dvou souborů, první soubor *.py* je modul napsaný v programovacím jazyku Python, druhým souborem je samotný playbook, ve kterém je uveden nový modul včetně jeho potřebného nastavení. Nástroj Ansible dále umožňuje vytváření tzv. rolí, ty umožňují rozdělit konfiguraci na dílčí části, to znamená je vytvořen hlavní soubor *.yml*, ve kterém jsou uvedeny dílčí konfigurace. Je možné tak místo jednoho dlouhého konfiguračního souboru vytvořit těchto souborů hned několik z toho jeden je hlavní, v něm jsou pak uvedeny odkazy na dílčí podsoubory a zároveň slouží ke spuštění dané konfigurace. [18], [19]

2.1.1 Instalace Ansible

Ansible bude nainstalován na klientský stroj, pomocí kterého se bude vytvářet, popřípadě upravovat konfigurace jednoho nebo i více serverů. Pro spojení bude možné využít protokol SSH a tím tak zajistit komunikaci klient – server.

Před začátkem instalace je nutné mít na obou zařízeních (klient, server) nainstalované potřebné balíčky a nejnovější aktualizace. Instalace všech těchto potřebných položek se na obou virtuálních systémech provede prostřednictvím terminálu a následujících příkazů

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python
```

První dva příkazy zajistí stažení dostupných balíčků a jejich následnou instalaci, poslední příkaz doinstaluje balíčky s programovacím jazykem Python.

Instalace samotného nástroje Ansible na klientský stroj je v případě operačního systému Ubuntu velmi jednoduchá, lze ji provést pomocí pouze jednoho příkazu

```
sudo apt-get install ansible
```

Tímto je instalace hotova a je možné přejít ke konfiguraci spojení mezi klientem (Ansible) a serverem. Ansible si uchovává záznam se všemi hosty, jedná se o textový soubor *hosts*. V tomto záznamu je nutné uvést IP adresu, popřípadě doménové jméno. K vytvoření SSH spojení v souboru *hosts* budou kromě adresy serveru přidány i parametry týkající se SSH spojení. Tato konfigurace bude provedena prostřednictvím terminálu a textového editoru (*nano* nebo *vi*) pomocí následujících příkazů

```
sudo nano /etc/ansible/hosts
```

Zde se přidají parametry daného serveru

```
[Název skupiny]
<IP adresa serveru>
ansible connection = ssh
ansible ssh user = <uživatelské jméno>
ansible ssh pass = <uživatelské heslo>
ansible sudo pass = <heslo pro sudo>
```

Tímto je vytvořen jednoduchý seznam hostů, je zde vytvářet i skupiny serverů a vytvořenou konfiguraci poslat několika serverům najednou. Kromě IP adresy serveru jsou uvedeny i přihlašovací údaje pro přihlášení prostřednictvím protokolu SSH. Ve většině případů bude nutné pro vykonání určité konfigurace, kde jsou různé instalace a přístupy do systémových složek serveru, a proto je nutné zajistit, aby bylo možné vykonávat konfiguraci prostřednictvím SSH jako *root*.

Tím je požadované spojení nastaveno a je možné ho ověřit zadáním následujícího příkazu

```
sudo ansible <IP serveru> -m ping
```

Tím bude ověřeno spojení s daným serverem, je však možné místo IP adresy serveru zadat *all* a ověřit spojení se všemi servery nebo je možné ověřit spojení se skupinou serverů zadáním názvu této skupiny. V případě, že spojení klient – server je úspěšné, obdržíme odpověď v podobě `SUCCES` a potvrzení `"ping":"pong"`. [20], [21]

2.2 Terraform

Terraform je další z nástrojů pro automatizaci infrastruktury patřící americké společnosti HashiCorp, která sídlí v San Franciscu ve státě Kalifornie. Společnost HashiCorp založil Mitchel Hashimoto a Armon Dadgar v roce 2012. Firma se specializuje na software s otevřeným zdrojovým kódem (open-source), který je zaměřen zejména na poskytování služeb. Dnes má společnost HashiCorp kolem jednoho tisíce zaměstnanců. Kromě nástroje Terraform poskytuje tato společnost další nástroje jako je například nástroj Vault pro ochranu, správu a šifrování dat, Consul pro správu a zabezpečení síťových služeb, Nomad pro správu aplikací,

Vagrand umožňující správu a vytváření virtuálních prostředí a nástroj Packer pro vytváření obrazů strojů. [22]

Nástroj Terraform byl poprvé představen v červenci roku 2014. Tento nástroj umožňuje prostřednictvím jednoho vytvořeného konfiguračního souboru vytvořit požadovanou strukturu infrastruktury. Konfigurační soubory je možné vytvářet buďto pomocí konfiguračního jazyka společnosti HashCorp (HCL) nebo pomocí objektového programovacího jazyku JavaScript Object Notation (JSON). Terraform podporuje různé poskytovatele služeb a jejich produkty mezi které patří například Microsoft Azure, OpenStack, Goole Cloud, VMware vSphere, Amazon Web Services a další. Terraform je nabízen hned v několika verzích, a to v základní klientské verzi CLI nebo rozšířenější Cloud. Verze CLI poskytuje správci prostředky pro přenos a provedení konfigurace popsané v konfiguračním souboru. Verze Cloud má navíc tzv. cloudovou vrstvu, která zajišťuje hned několik rolí, mezi které patří: správa konfiguračních souborů a jejich následné provedení, dále poskytuje aplikační rozhraní pro uživatelé. Obecně Terraform uživateli nabízí možnost vytvořit infrastrukturu tak, že nejdříve kódem popíše, jak by si ji představoval, například chce vytvořit server, následně Terraform vyhledá všechny změny a zjistí, zda daný server existuje či ne, pokud neexistuje, je následně vytvořen. Terraform je možné nainstalovat na operační systémy Linux, Windows, macOS, FreeBSD či OpenBSD. Využití nástroje Terraform závisí na již zmíněných poskytovatelích. Je možné využít jak základní provider, tak poskytovatele pro různé služby tzv. provisioner. Mezi základní patří například *file Provisioner* pro práci se soubory, *remote-exec Provisioner* pro zadávání příkazů, *puppet Provisioner* pro management prostřednictvím nástroje Puppet. Terraform je však využíván k vytváření konfigurací v rámci různých služeb jako jsou například cloudové služby, konfigurace databází či serverů. [23]

2.2.1 Instalace Terraform

Před začátkem instalace je nutné ověřit, zda operační systém, na který bude tento nástroj později instalován má nainstalovány všechny dostupné aktualizace a má nainstalován nástroj Unzip pro extrahování *.zip* souborů, popřípadě je nutné tento balíček doinstalovat například prostřednictvím terminálu a příkazu `sudo apt-get install unzip`. Pokud operační systém obsahuje všechny zmíněné náležitosti, je možné přejít k instalaci nástroje Terraform.

Nejdříve je nutné přejít na stránky Terraform a zde přejít do sekce pro stažení konkrétně *Download CLI*. Zde je nutné si vybrat verzi pro konkrétní systém a následně balíček ze stránek nástroje stáhnout, to je možné provést v případě desktopové verze operačního systému pomocí internetového prohlížeče nebo například pomocí terminálu a příkazu

`wget https://releases.hashicorp.com/terraform/0.12.12/`
terraform_0.12.12_linux_amd64.zip
Po stažení, je potřeba tento soubor rozbalit, k tomu poslouží již zmíněný nástroj Unzip. Rozbalení souboru se provede zadáním příkazu
`unzip terraform_0.12.12_linux_amd64.zip`
V tomto adresáři by se měl nacházet jeden spouštěcí soubor. Dalším krokem je vytvoření tzv. pracovního adresáře, do kterého se budou vkládat a následně spouštět jednotlivé konfigurační soubory (*.tf*), to se provede následně
`mkdir <jméno pracovního adresáře>`
Nyní je nutné zajistit, aby bylo mužné spouštět příkazy nástroje Terraform z této pracovní složky, proto je nutné nejdříve do tohoto adresáře vstoupit
`cd <jméno pracovního adresáře>/`
a zde zadat následující příkaz
`echo $"export PATH=\$PATH:$(pwd)" >> ~/ .bash_profile`
Tím bude zajištěno, že pracovní adresář bude pomocí proměnné *PATH* přidán na seznam k hledání spustitelných souborů, v tomto případě *./bash_profile*, je potřeba tento seznam aplikovat, to je docíleno příkazem
`source ~/ .bash_profile`
Nyní je nástroj Terraform připraven k použití. V případě serveru, který bude nástrojem Terraform konfigurován je nutné nastavit spojení prostřednictvím protokolu SSH, aby se klient k serveru mohl přihlásit jako *root*, a mohl tak využívat oprávnění *sudo*. Správné nastavení SSH spojení je pak zajištěno na straně serveru a úpravou konfiguračního souboru protokolu SSH následujícími příkazy
`sudo nano /etc/ssh/sshf_config`
Zde se upraví následující: `#PermitRootLogin prohibit-password` na `PermitRootLogin yes.`
Nakonec se zadá heslo pro *root* následujícím příkazem
`sudo passwd root` [24]

2.3 Puppet

Puppet, který patří pod stejnojmennou společnost Puppet, byla založena roku 2005 v americkém městě Portland ve státě Oregon. Tato společnost se zabývá především vývojem nástrojů pro automatizaci infrastruktur, to znamená, že poskytuje služby pro konfiguraci, správu pro datová centra či cloudové služby. Společnost Puppet nabízí hned několik verzí jejich systému pro automatizaci infrastruktury například Puppet Enterprise nebo verzi s názvem Open Source Puppet. Rozdíl je v tom, že verze Enterprise nabízí větší množství funkcí a je především učena pro správu větších organizací, zatímco základní verze je určena pro menší podniky a zároveň se jedná o nástroj s otevřeným zdrojovým kódem. [25], [26]

Nástroj Puppet pro vytváření konfigurací poskytuje svůj deklarativní programovací jazyk, definujeme tak, co se má stát. Jedná se zde o komunikaci typu klient – server, která ale oproti ostatním zmíněným nástrojům funguje trochu odlišně. Systém, na kterém se vytváří jednotlivé konfigurace se nazývá master, konfigurované systémy jsou pojmenovány jako agent. Spojení master – agent, není založeno na protokolu SSH jako v předchozích případech, ale na vytváření a potvrzování certifikátů. [27]

2.3.1 Instalace Puppet

Instalace tohoto nástroje bude úplně odlišná od předchozích nástrojů, zatímco nástroje Ansible či Terraform jsou spíše založeny na základě spouštěcích souborů či balíčku, které je nutné nakonfigurovat, v případě nástroje Puppet se bude jednat o instalaci konkrétního software. Instalace tak bude rozdělena na tři části, v první části bude nutné nainstalovat tzv. Puppet Master, což je část nástroje, která se instaluje na klientský systém, přes který se bude provádět správa a konfigurace jednotlivých klientů. V druhé části se bude instalovat tzv. Puppet Agent na jednotlivé servery, které se budou konfigurovat prostřednictvím klienta s nainstalovaným Puppet Master. Poslední částí je zajištění samotné komunikace klient – server, která zde bude fungovat oproti předchozím nástrojům Ansible, Terraform trochu odlišně, bude zde použita komunikace na základě vytvoření a potvrzení certifikátu.

Před instalací nástroje Puppet je nutné u obou systémů (klient – server) do souboru *hosts* zapsat IP adresy klienta a serveru, což je nutné pro využití budoucí komunikace. Do souboru hosts je možné přejít prostřednictvím příkazu

```
sudo nano /etc/hosts
```

Zde se následně přidají jednotlivé IP adresy a názvy, pod kterými budou systémy dostupné

```
<IP adresa klienta> puppet
<IP adresa serveru> agent
```

Tím je dokončena konfigurace hostů a je možné tak přejít k samotné instalaci jednotlivých částí nástroje Puppet. [28]

Puppet Master

Instalaci mastera na klientský systém je možné provést prostřednictvím příkazu

```
sudo apt-get install puppetmaster
```

Po dokončení instalace je nutné provést konfiguraci mastera v souboru *puppet.conf*

```
sudo nano /etc/puppet/puppet.conf
```

Zde je nutné ověřit, popřípadě doplnit řádek se jménem hostitele, ke kterému se budou jednotliví agenti připojovat, jedná se o jméno, které bylo již uvedeno na obou systémech v souboru *hosts*.

```
dns_alt_names = puppet
```
Tímto je instalace a konfigurace klienta (Puppet Master) dokončena. Aby bylo možné vytvářet nové moduly a spouštět je prostřednictvím Puppet Agent, je nutné stáhnout a doinstalovat potřebný balíček, to se provede příkazem
```
wget https://apt.puppetlabs.com/puppet-release-bionic.deb
```
Instalace se provede pouhým rozbalením tohoto balíčku
```
sudo dpkg -i puppet-release-bionic.deb
```
Nakonec je potřeba vytvořit cílovou složku, ve které se budou později vytvářet projekty (moduly), tuto složku je možné vytvořit prostřednictvím příkazu
```
sudo mkdir -p
 /etc/puppet/code/environments/production/manifests/
```
[28], [29], [30]

Puppet Agent

Instalaci agenta na serverovém systému je možné provést obdobným způsobem jako v případě instalace mastera, a to prostřednictvím příkazu
```
sudo apt-get install puppet
```
Po dokončení instalace je však ještě nutné provést konfiguraci tohoto nainstalovaného agenta, postup konfigurace je stejný jako v případě mastera, a to úpravou souboru *puppet.conf*, zde bude nutné přidat následující
```
server = puppet
```
V rámci nástroje Puppet je tak nutné přidání tohoto řádku provést například do již předdefinované části `[master]`. Tím je konfigurace a instalace serveru (Puppet Agent) dokončena. [28], [29], [30]

Vytvoření certifikátu

Poslední částí instalace je vytvoření spojení klient – server prostřednictvím certifikátu. V rámci systému s nainstalovaným Puppet Agent se manuálně vytvoří SSL klíč, to se provede příkazem pro navázání spojení s Puppet Master
```
sudo puppet agent --test
```
Spojení však ještě nebude navázáno, ale dojde k vygenerování klíče. Nyní na systému s nainstalovaným Puppet Master je možné si zobrazit seznam nevyřízených žádostí o spojení
```
sudo puppet cert list
```
Zde by měla být vidět nevyřízená žádost o spojení s vygenerovaným certifikátem, následně je možné provést připojení prostřednictvím následujícího příkazu
```
sudo puppet cert sign <Jméno agenta>
```
Je však možné místo jména konkrétního agenta se připojit ke všem dostupným agentům, místo `<Jméno agenta>` se tak zadá `all`. [28], [30]

3 VYTVÁŘENÍ KONFIGURACÍ PRO ŘÍZENÍ INFRASTRUKTURY

Tato kapitola bude zaměřena na vytváření konfiguračních souborů prostřednictvím nástrojů pro konfiguraci infrastruktur, které byly zmíněny v předchozí kapitole, a to Ansible, Terraform či Puppet. V rámci této kapitoly bude na jednoduchém příkladu demonstrována konfigurace serveru prostřednictvím jednotlivých nástrojů. Jako ukázka zde bude použita instalace a jednoduchá konfigurace Nginx serveru. Nginx je webový a proxy server, který pracuje s protokoly HTTP, POP3, IMAP, SMTP a mnoha dalšími. První verze tohoto serveru se objevila kolem roku 2004, tento server je využíván různými firmami jako jsou: Česká Spořitelna, WordPress, Nokia, Seznam.cz, Dropbox nebo i některé mediální společnosti například FTV Prima. Realizace tohoto serveru bude probíhat tak, že bude vytvořen samotný konfigurační soubor pro jednotlivé nástroje. V tomto souboru se budou nacházet jednotlivé příkazy pro instalaci a konfiguraci Nginx serveru nebo konfigurace hosta. Dále bude vytvořen konfigurační soubor Nginx serveru, a nakonec bude vytvořen soubor *.html* (webová stránka). Tato ukázka tak bude vhodná zejména pro porovnání jednotlivých nástrojů, zmapování jejich kladů a záporů, popřípadě náročnosti na vytvoření a provedení samotné konfigurace. [31], [32]

3.1 Instalace Nginx serveru prostřednictvím Ansible

Před vytvořením konfiguračního souboru pro instalaci a konfiguraci Nginx serveru je nejdříve nutné přejít do adresáře nástroje Ansible

```
cd /etc/ansible
```

V tomto adresáři se vytvoří soubor, který bude pojmenován například *nginx.yml* prostřednictvím příkazu

```
sudo nano nginx.yml
```

Jelikož Ansible využívá pro konfiguraci jazyk YAML mají soubory příponu *.yml*, těmto souborům se říká scénáře, popřípadě playbook, instrukce v tomto souboru jsou vykonávány postupně instrukce po instrukci. Struktura souboru *.yml*: používají se pouze pomlčky, tabulátor nelze použít, dodržuje se odsazování, začátek nějaké části (oddílu) začíná pomlčkou. Zdrojový kód *nginx.yml* je zobrazen jako Soubor 1. První část tohoto kódu v tomto případě konfigurace hosta je věnována nastavení hosta či hostů, je zde definována IP adresa hosta a je mu umožněno využívat oprávnění *root*. Druhá část, v tomto případě *tasks* definuje úkoly, které budou postupně za sebou vykonávány, v této situaci se jedná o pět úkolů. Prvním úkolem je samotná instalace serveru Nginx prostřednictvím metody *apt*, která stáhne instalační balíček, `update_cache=yes` zajišťuje, aby mezipaměť *apt*

modulu byla vždy před nějakými změnami aktualizována. Druhým úkolem je nakopírování konfiguračního souboru *default* hostovi prostřednictvím metody *copy* pod názvem *default*. Třetím úkolem je následné spuštění nakopírované konfigurace. Předposledním úkolem je vytvořené webové stránky (*index.html*) nakopírovat hostovi a následně je proveden poslední úkol této části scénáře, a to je samotný restart Nginx serveru po jeho předchozí konfiguraci. [33]

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: instalace nginx
      apt: name=nginx update_cache=yes

    - name: nakopirovani nginx konfigurace
      copy: src=files/default
            dest=/etc/nginx/sites-enabled/default

    - name: nakopirovani html stranek
      template: src=templates/index.html.j2
                dest=/usr/share/nginx/html/index.html
                mode=0644

    - name: restart nginx
      service: name=nginx state=restarted
```

Soubor 1 Konfigurační soubor *nginx.ylm* nástroje Ansible

Druhý soubor, který bylo nutné vytvořit je *default*, na kterém je uložena konfigurace samotného Nginx serveru, kde je možné nastavit například výchozí naslouchání na portu 80, dále je zde nutné nastavit cestu ke skriptu s webovými stránkami, formát těchto stránek (například *.html* či *.php*), nastavení názvu serveru, to znamená, prostřednictvím jakého názvu se dostaneme na webové stránky spuštěné na tomto serveru, nastavení proti zneužití a mnoho dalších nastavení. Konfigurační soubor *default* je zobrazen jako Soubor 2.

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /usr/share/nginx/html;
        index index.html index.htm;

        server_name localhost;
```

```
        location / {
                try_files $uri $uri/ =404;
        }
}
```

Soubor 2 Nastavení Nginx serveru *default*

Třetím a zároveň posledním je soubor *index.html*. Jedná se o skript napsaný ve značkovacím jazyku HTML. Byla zde tak vytvořena jednoduchá webová stránka s nadpisem a textem pro demonstrační účely, pro zajištění správné diakritiky zde bylo ještě doplněno UTF-8 kódování.

Po vytvoření všech konfiguračních souborů je možné na hosta s danou IP adresou nainstalovat Nginx server, to se provede tak, že na systému s nainstalovaným nástrojem Ansible, se zadá následující příkaz

```
sudo ansible-playbook nginx.yml
```

Funkčnost může být ověřena na klientském systému, kde se otevře webový prohlížeč a zadá se IP adresa serveru. Ukázka otevřené webové stránky (*index.html*) je zobrazena jako Obrázek 1. [33]

Obrázek 1 Zkušební webová stránka *index.html*

3.2 Instalace Nginx serveru prostřednictvím Terraform

Veškerá konfigurace bude probíhat v pracovním adresáři, který byl vytvořen v rámci kapitoly 2.2.1. V tomto adresáři je nutné vložit, popřípadě vytvořit například pomoci editoru *nano* konfiguraci Nginx serveru *default* a zkušební webové stránky *index.html*. Oba tyto soubory byly popsány v kapitole 3.1. Dále se zde vytvoří konfigurační soubor nástroje Terraform, konkrétně *nginx.tf*, který je zobrazen jako Soubor 3. Tento soubor je možné vytvořit například pomocí editoru *nano* prostřednictvím následujícího příkazu

```
sudo nano nginx.tf
```

```
resource "null_resource" "nginx" {

# Instalace balicku
 provisioner "remote-exec" {
    inline = [
      "apt-get update",
      "apt-get -y upgrade",
      "apt-get -y install nginx",
      "systemctl start nginx"
    ]
  }

# Nakopirovani konfigurace
 provisioner "file" {
   source      = "/home/klient/terraform-project/default"
   destination = "/etc/nginx/sites-enabled/default"
 }


# Nakopirovani html
 provisioner "file" {
   source      = "/home/klient/terraform-project/index.html"
   destination = "/usr/share/nginx/html/index.html"
 }

# Nastaveni spojeni
 connection {
    type     = "ssh"
    user     = "root"
    password = "<heslo pro root>"
    host     = "<IP serveru>"
  }

}
```

Soubor 3 Konfigurační soubor *nginx.tf* nástroje Terraform

Samotný konfigurační soubor se skládá ze čtyř částí. Nejdříve dojde k instalaci aktualizací a samotného nástroje Terraform. Druhým a třetím krokem je nakopírování konfigurace Nginx serveru a zkušebních webových stránek do potřebných adresářů. Poslední částí je nastavení spojení a konfigurace hosta.

Po vytvoření všech potřebných souborů, je možné přejít k realizaci vzdálené konfigurace. Před vytvořením spojení se nejdříve provede tzv. inicializace, to znamená, že nástroj Terraform prohledá pracovní adresář a vyhledá všechny doposud neprovedené konfigurace, zároveň dojde ke zkompilování těchto konfigurací. [34], [35] Inicializace se provede zadáním následujícího příkazu

```
terraform init
```

V případě, že konfigurace byla nalezena, zkontrolována a v samotném kódu nebyly nalezeny žádné chyby je možné přejít ke spuštění samotné konfigurace serveru. Terraform nabízí možnost si zobrazit plán konfigurace, to je vhodné

zejména pro zkontrolování toho, zda bude konfigurace se vykonána, jak má, této plán je možné zobrazit příkazem
`terraform plan`
Konfigurace se pak spustí následně
`terraform apply` [24]

3.3 Instalace Nginx serveru prostřednictvím Puppet

Při konfiguraci serveru prostřednictvím nástroje Puppet mluvíme o vytvoření tzv. modulu, v případě modulu však hovoříme o složce. V této složce se vytvoří soubor typu *.pp*, který zde funguje jako konfigurační, dále se do této složky vytváří další podsložky, například s dalšími soubory, které se budou například kopírovat do serveru. Pro instalaci a konfiguraci Nginx serveru však postačí vytvořit pouze konfigurační soubor.

Před vytvořením nové konfigurace je nutné nejdříve přejít do pracovního adresáře klienta s nainstalovaným Puppet master, který byl vytvořen v rámci kapitoly 2.3.1, do tohoto adresáře se přejde prostřednictvím příkazu
`cd /etc/puppet/code/environments/production/manifests/`
V tomto adresáři se vytvoří konfigurační soubor s názvem *init.pp*
`sudo nano init.pp` [30]
Zdrojový kód tohoto souboru je zobrazen jako Soubor 4.

```
#Instalace Nginx serveru
package { 'nginx':
  ensure => installed,
}

#Nakopirovani nginx.conf
file { '/etc/nginx/sites-enabled/default':
  ensure => file,
  content => 'server {
                        listen 80 default_server;
                        listen [::]:80 default_server;

                        root /usr/share/nginx/html;
                        index index.html index.htm;

                        server_name localhost;

                        location / {
                                try_files $uri $uri/ =404;
                        }
                      }',
```

```
}

#Nakopirovani index.html
file { '/usr/share/nginx/html/index.html':
  ensure => file,
  content =>'<!DOCTYPE html>
            <html>
            <head>
            <title>Ansible page</title>
            </head>
            <meta charset="UTF-8">
            <body>

            <h1>Automatická konfigurace virtuální infrastruktury</h1>
            <p>Zkušební stránka</p>

            </body>
            </html>',
}
```

Soubor 4 Konfigurační soubor *init.pp* nástroje Puppet

Zdrojový kód *init.pp* je rozdělen do tří částí, kde první část zajišťuje instalaci samotného Nginx serveru, další dvě části pak zajišťují vytvoření souborů *default* a *index.html* v cílových adresářích. Oba tyto soubory byly popsány v kapitole 3.1. [36]

Konfiguraci je nutné spustit prostřednictvím Puppet agent, jak již bylo zmíněno v kapitole 2.3.1 prostřednictvím příkazu

```
sudo puppet agent --test
```

Tímto je instalace a konfigurace Nginx serveru dokončena, je však nutné buďto Nginx server nebo celý systém s Puppet agent restartovat, poté je možné ověřit funkčnost otevřením webového prohlížeče a zadáním IP adresy serveru, měla by se zobrazit webová stránka, ta je zobrazena jako Obrázek 1

4 POROVNÁNÍ DOSTUPNÝCH NÁSTROJŮ PRO SPRÁVU VIRTUÁLNÍ INFRASTRUKTURY

Tato kapitola bude věnována zhodnocení dosavadní práce s nástroji, které umožňují vzdálené řízení a správu virtuální infrastruktury, budou zde zmíněny jejich výhody, popřípadě nevýhody. Budou zde uvedeny všechny poznatky a zkušenosti z instalací, popřípadě vytváření spojení prostřednictvím jednotlivých nástrojů. Dále bude tato kapitola věnována zhodnocení těchto nástrojů z hlediska vytváření či spouštění konfigurací pro správu a řízení serveru, popřípadě serverů. Na základě těchto kladů či záporů bude nakonec určen nejvhodnější nástroj ke správě a řízení infrastruktury, který bude dále využíván v rámci této práce.

4.1 Ansible

V případě nástroje Ansible je jeho instalace vcelku jednoduchá, jelikož je možné ji provést prostřednictvím balíkovacího systému *apt*, to znamená, že je možné prostřednictvím jednoho příkazu v terminálu stáhnout a nainstalovat balíček s tímto nástrojem se všemi potřebnými prvky a nástroji. Hardwarové nároky se odvíjí od různých faktorů, zejména však počtu konfigurovaných serverů, kdy konfigurace může probíhat na všech serverech současné. V případě uložiště samotná instalace není náročná, u instalačního balíčku mluvíme o velikosti 78,3 MB. Velkou výhodou Ansible je instalace pouze v rámci klienta. Nutností je však mít jak v případě klienta, tak serveru nainstalován balíček se skriptovacím jazykem Python. Ansible vyžaduje konfiguraci hostů v samotném souboru *hosts*, zde je možné přidávat buďto samotné hosty nebo vytvářet skupiny hostů o různém počtu. Pro zajištění funkčnosti spojení a potřebného oprávnění je nutné tyto hosty správně definovat, to znamená je nutné uvést parametry jako je například druh spojení, uživatelské jméno, hesla, popřípadě heslo k uživateli *root*. Pokud budeme hovořit o samotných konfiguracích (playbooky) pro dané servery, je nutné dodržovat správnou syntaxi jazyka YAML, může se stát, že samotné provádění konfigurace (playbooku) daného serveru může být přerušeno kvůli nějaké chybě v syntaxi, může se jednat o špatné odsazení, chybějící znaky jako je například pomlčka, dvojtečka, popřípadě chybějící nebo přebytečné mezery. Chyby v mezerách mohou být způsobeny i při odstraňování nějakého modulu, po jeho odstranění pak mohou vzniknout chyby v ostatních modulech a je třeba mezery odstranit a znovu vytvořit. Je však možné spustit kontrolní režim, kdy je konfigurace spuštěna, nedochází však ke změnám na straně konfigurovaného serveru, tím je možné před spuštěním nejdříve danou konfiguraci ověřit a na základě výsledků ji pak opravit, kontrola se provede zadáním příkazu

```
sudo ansible-playbook <jméno souboru>.yml --check
```

Při spuštěné konfiguraci Ansible nijak nezobrazuje průběh právě prováděných činností, to lze částečně vyřešit přidáváním komentářů k jednotlivým částem kódu, tyto komentáře jsou pak při spuštěné konfiguraci vidět a uživatel má alespoň přehled o tom, která část jeho playbooku je právě vykonávána. Výhodou Ansible může být i to, že kód napsaného konfiguračního souboru je vykonán postupně, to je vhodné zejména při instalaci a konfiguraci nějaké serverové služby například Nginx serveru, kdy po jeho instalaci a konfiguraci je nutný restart, tímto bylo tak zajištěno, že se Nginx nejprve nainstaloval, nakonfiguroval a nakonec restartoval. Mezi další přednosti nástroje Ansible patří obsáhlá dokumentace výrobce, je zde popsána instalace, konfigurace nebo vytváření modulů.

4.2 Terraform

Instalace nástroje Terraform jak již bylo zmíněno v kapitole 2.2.1 je rozdělena do několika kroků, nejdříve je nutné stáhnout *.zip* soubor o velikosti 16,4 MB. Po jeho rozbalení získáme spustitelný soubor o velikosti 50,8 MB. Dále se vytvoří pracovní adresář a zajistit, aby bylo možné Terraform spustit v pracovním adresáři. I když je instalace spíše manuální, není nutné doinstalovat žádné další balíčky. Hardwarové nároky na klienta jsou stejně jako v případě nástroje Ansible závislé především na počtu konfigurovaných serverů. V rámci nástroje Terrfaorm není nutné na straně klienta provádět konfiguraci hostů (serverů), ale pro zajištění funkčnosti SSH spojení, konkrétně zajistit možnost se připojit s oprávněním *root*, je nutné na straně konfigurovaného serveru mít správně nastavenu službu SSH a heslo pro správce *root*. I v případě nástroje Terraform je nutné dodržovat syntaxi konkrétně jazyku HCL, není zde však kladen na syntaxi takový detailní důraz jako v případě nástroje Ansible a jazyku YAML, není zde brán takový důraz na úplně přesnou pozici (odsazení) nebo použití určitého počtu mezer. Uživatel má celkově lepší přehled nad vykonávanou konfigurací, před jejím začátkem je možné si zobrazit plán, kdy Terraform vyhledá změny a zobrazí, které úkony se po spuštění konfigurace vykonají. Po spuštění samotné konfigurace jsou zobrazeny všechny úkony a instrukce, které se momentálně vykonávají a je tak možné lépe detekovat chybu, která není v samotné syntaxi, ale například v příkazu. V dokumentaci nástroje Terraform je možné kromě několika základních modulů najít tzv. poskytovatele, pro které je možné vytvářet konfigurace.

4.3 Puppet

Samotná instalace nástroje Puppet byla oproti zmíněným dvěma nástrojům rozsáhlejší, před instalací je nutné přidat IP adresy a doménová jména klienta

a serveru do souboru operačního systému *hosts*, a to jak v případě klienta, tak serveru. Instalace nástroje Puppet byla provedena stažením a nainstalováním balíčku o velikosti 54,6 MB. Po instalaci tohoto balíčku je nutné ještě provést konfiguraci klienta (Puppet Master) úpravou souboru *puppet.conf*, kde bylo nutné zadat doménové jméno, pod kterým klient bude dostupný. Dále je nutné v rámci klienta doinstalovat balíček, který zajišťuje možnost vytváření nových modulů. V případě serveru je nutné doinstalovat tzv. agenta, bude tak nutné stáhnout a nainstalovat balíček o velikosti 54,5 MB. Dále je potřeba provést konfiguraci úpravou souboru *puppet.conf* a zde přidat jméno klienta. Posledním krokem je zajištění spojení, to se provede prostřednictvím výměny certifikátu klient – server. V případě vytváření nových modulů je nutné dodržet cestu k pracovnímu adresáři, která je však oproti ostatním nástrojům delší, aby bylo možné daný modul spustit. Hardwarové nároky jsou i zde závislé hlavně počtu konfigurovaných serverů. Jednotlivé dílčí moduly ve vytvářeném manifestu mají kratší zápis, ale v některých případech horší strukturu, kdy v případě nakopírování souboru na určité místo bylo jednodušší použít modul, který soubory v tomto místě vytvoří a zapíše do nich konkrétní atributy. V případě spuštění konfigurace nástroj Puppet zobrazuje informace během přenosu, jako je například informování o již provedených akcích, zobrazení upozornění v případě provedení změn, zda proběhly úspěšně, popřípadě v jakém čase.

5 VÝBĚR A VYUŽITÍ VHODNÉHO NÁSTROJE

Na základě předchozích kapitol bude cílem této kapitoly vybrat konkrétní nástroj pro konfiguraci virtuální infrastruktury, který bude použit v dalších částech této práce. Na základě teoretických poznatků, zkušeností a výsledků z předchozích kapitol se jeví jako nejvhodnější nástroje Ansible a Terraform. Oproti nástroji Puppet mají podstatnou výhodu zejména v rámci instalace, kdy není nutné provádět instalaci na straně konfigurovaného serveru, tím je samotná vzdálená konfigurace mnohem efektivnější. Cílem této práce je však vybrat jeden nástroj, který se jeví jako nejuniverzálnější, to znamená, že kromě instalací jednotlivých aplikací nebo dílčích konfigurací dokáže vytvořit nějakou infrastrukturu, popřípadě služby.

5.1 Instalace a konfigurace WordPress

V případě nástroje Wordpress se jedná o publikační nástroj pro tvorbu zejména webových stránek či blogů. Wordpress je napsán v programovacím jazyku PHP a je dostupný jako open-source. Cílem bude nainstalovat a nakonfigurovat tento publikační software, včetně MySQL databáze, která bude sloužit jako databáze uživatelů a databáze projektů a Nginx serveru, na kterém bude běžet samotný WordPress prostřednictvím nástrojů Ansible a Terraform.

V případě MySQL mluvíme o tzv. multiplatformní databázi (Linux, Windows a další), která je ovládána a konfigurována prostřednictvím strukturovacího jazyku SQL. MySQL momentálně spadá pod společnost Oracle Corporation. Databáze je v současnosti dostupná pod dvěma licencemi, a to bezplatná GNU a komerční.

I když nástroje nabízejí různé moduly, při instalaci budou využity zejména ty základní moduly k vyzkoušení funkčnosti obou nástrojů.

5.1.1 Instalace WordPress prostřednictvím Ansible

V rámci této konfigurace byl vytvořen konfigurační soubor *wordpress.yml*, ve kterém jsou dílčí kroky instalace a konfigurace systému WordPress. Konfigurace wordpress.yml je zobrazena jako Soubor 5.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: instalace nginx
      apt: name=nginx update_cache=yes

    - name: nakopirovani nginx konfigurace
      copy: src=files/default
            dest=/etc/nginx/sites-enabled/default
```

```
    - name: stazeni a extrahovani Wordpress
      unarchive:
        src: https://cs.wordpress.org/latest-cs_CZ.tar.gz
        dest: /home/server
        remote_src: yes

    - name: restart nginx
      service: name=nginx state=restarted

    - name: instalace php
      apt:
        pkg:
        - php-fpm
        - php-mysql

    - name: vlozeni prikazu do php.ini
      lineinfile:
        path: /etc/php/7.3/fpm/php.ini
        line: cgi.fix_pathinfo=0

    - name: restart php-fpm
      systemd:
        state: restarted
        name: php7.3-fpm

    - name: instalace Mysql
      apt:
        name: mysql-server

    - name: konfigurace Mysql a vytvoreni databaze wordpress
      command: 'sudo mysql -e "{{item}}"'
      with_items:
        - ALTER USER 'root'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'password';

    - name: vytvoreni databaze
      command: 'mysql -u root -ppassword -e "{{item}}"'
      with_items:
        - FLUSH PRIVILEGES;
        - CREATE DATABASE wordpress;

    - name: nakopirovani konfigurace wp-config.php
      copy: src=files/wp-config.php
            dest=/home/server/wp-config.php
```

Soubor 5 Instalace a konfigurace WordPress prostřednictvím Ansible

Jak již bylo zmíněno Ansible, konkrétně jazyk YAML jde instrukci po instrukci, proto byl nejdříve nainstalován Nginx server, do kterého se dále nakopírovala konfigurace *default*, byl stažen a rozbalen WordPress, nakonec je server restartován. Následně se provedla instalace balíčku programovacího jazyku PHP, konkrétně spouštěč procesů *php-fpm* a *php-mysql*. Po instalaci je nutné do souboru *php.ini*

z důvodu zabezpečení přidat řádek `cgi.fix_pathinfo=0` a poté spouštěč procesů *php-fpm* restartovat. Tím je instalace WordPress hotova, aby bylo možné tento publikační systém naplno používat je nutné instalovat a konfigurovat MySQL databázi. V této části se však objevilo několik komplikací, v první řadě vytvořit uživatele, který bude mít přístup do vytvořené databáze, kdy po jejich vytvoření se nebylo možné v rámci WordPress přihlásit, nakonec tento problém byl vyřešen příkazem `ALTER USER`, který umožnil resetovat heslo k účtu *root*, což k této demonstraci plně postačovalo. Druhý problém byl již v rámci nástroje Ansible, kde bylo nutné vyřešit příkazy, u kterých bylo nutné zadávat další parametry jako je například uživatelské jméno či heslo. I když samotný Ansible poskytuje různé moduly, kdy je možné heslo zadat, ale v rámci konfigurace databáze je nutné spustit její příkazovou řádku a v ní provádět jednotlivou konfiguraci. Z tohoto důvodu bylo jednodušší využít volbu *execute* (`-e`) databáze MySQL. Tato volba umožní spustit konfiguraci databáze, vykonat požadovaný příkaz a konfiguraci ukončit. Nakonec je potřeba nakopírovat konfiguraci systému WordPress do jeho pracovního adresáře, který byl stažen a rozbalen v předcházejících krocích, v této konfiguraci se nachází zejména přihlašovací údaje k databázi a její jméno a další potřebná data. Nyní je instalace hotova a je možné založit nový projekt. [37], [38], [39]

5.1.2 Instalace WordPress prostřednictvím Terraform

Konfigurace se stejnou strukturou byla vytvořena i pomocí nástroje Terraform, konkrétně *wordpress.tf*. Terraform nenabízí modul k instalaci balíčků, proto je využit modul pro zadávání příkazů, zároveň je nutné ošetřit příkazy, kde je nutné potvrzování nebo zadávání textového řetězce jako je například uživatelské jméno či heslo, při spuštěné konfiguraci nelze nic zadat a konfigurace se v tomto bodě zastaví. U příkazu pro instalaci balíčků, kde je potřeba provést potvrzení je nutné zadat `-y`, tím je zajištěno potvrzení a instalace pokračuje. Konfigurace však v tomto případě nebyla úspěšně dokončena, problém nastal v konfiguraci MySQL databáze při resetování hesla příkazem `ALTER USER`, kdy došlo ke kolizi syntaxí. Při resetování zde byl použit modul pro zadávání příkazů, kdy se mezi dvojité uvozovky zapíše požadovaný příkaz, ale příkaz pro reset hesla, kdy je zároveň použita volba *execute* využívá také dvojité uvozovky, mezi které se zadá příkaz pro MySQL databázi. Terrafom při spuštění konfigurace zahlásí chybu, kde je uvedeno, že v této části kódu nejsou tyto uvozovky očekávány. Problém by se dal ošetřit použitím jednoduchých uvozovek u příkazů pro MySQL databázi, což i fungovalo, ale byl zde problém v případě příkazu pro resetování hesla databáze, kdy nové heslo je uvedeno také v jednoduchých uvozovkách a tento příkaz nebyl z důvodu chyby syntaxe pro použitou verzi MySQL databáze proveden a nebylo možné se v nástroji WordPress k této databázi připojit. [34], [35], [38], [39]

6 POUŽITÉ SLUŽBY A PROTOKOLY

Tato kapitola se bude zabývat službami a protokoly, které budou použity při realizaci serveru, kde tyto jednotlivé služby a protokoly budou spuštěny v rámci jednotlivých kontejnerů. Budou vytvořeny čtyři kontejnery se službami, a to FTP, Nginx, Apache a Lighttpd kontejner. Z hlediska protokolů zde bude použit FTP protokol, v případě Nginx, Apache, lighttpd pak HTTP.

6.1 FTP protokol

Protokol FTP je určen pro přenos dat mezi dvěma stroji, jedná se o komunikaci klient server, kdy klient může přistupovat k jednotlivým souborům, to znamená může je například kopírovat, odstraňovat či přejmenovat. V případě FTP se jedná o jeden ze starších protokolů, kdy první konfigurace tohoto protokolu se datuje kolem roku 1971. Za tuto dobu byl protokol několikrát modifikován například pro podporu IPv6. FTP pro přenos využívá TCP protokol a dva porty. První port 21 je používán pro příkazy a odpovědi, druhý port 20 je využíván již pouze pro samotný přenos dat. V případě FTP protokolu je možné samotné připojení realizovat prostřednictvím dvou režimů, konkrétně aktivního či pasivního režimu. [40]

Aktivní režim
V případě tohoto režimu klient zažádá server o spojení, to znamená vyšle příkaz serveru na port 21, server mu odpoví nově vytvořeným spojením na portu 20 pro přenos dat. [40], [41]

Pasivní režim
Problém však nastává v případě, kdy je použit překladač adres NAT nebo brána Firewall mezi klientem a serverem. V tomto případě není možné, aby server vytvořil spojení z portu 21 na některý z portů klienta. Klient v tomto případě zažádá o pojení se serverem a zašle příkaz `PASV`, který informuje server o použití pasivního režimu. Server zasílá zpět klientovi adresu a port, na které bude naslouchat, klient tak může realizovat spojení pro přenos dat. [40], [41]

FTP protokol umožňuje i několik možností přihlášení, a to pomocí uživatelského jména a hesla nebo pomocí tzv. anonymního režimu, kdy klient nemusí zadávat přihlašovací údaje, je automaticky přihlášen v režimu hosta, zároveň je mu omezen přístup v rámci adresářů.

Pro přístup klienta do FTP serveru je možné použít například webový prohlížeč nebo průzkumník operačního systému a zadat následující příkaz `ftp://<uživatelské jméno>:<heslo>@<IP adresa serveru>`

či terminál (příkazový řádek). V případě terminálu se zadá příkaz `FTP`, tím je spuštěn FTP-klient, následně příkazem `open` je možné otevřít spojení, klient je následně vyzván k zadání adresy serveru a jeho přihlašovacím údajům. Další možností je využití například softwaru FileZilla či správce souborů Total Commander.

V případě výhod a nevýhod samotného FTP protokolu patří mezi jeho výhody zejména možnost využití aktivního a pasivního režimu, dále jeho jednoduchost například vůči HTTP protokolu. S tím je však spojena hlavní nevýhoda tohoto protokolu, tou je samotná bezpečnost. FTP protokol není nijak zabezpečen či šifrován, je možné tak zachytávat pakety, ze kterých je možné zjistit například uživatelské jméno či heslo potřebné pro přihlášení k FTP serveru.

vsFTPD server

Vsftpd server jednoduchý, bezpečný a stabilní FTP server pro operační systémy UNIX, je tak například součástí operačních systémů CentOS nebo Fedora. Veškerá konfigurace serveru probíhá prostřednictvím konfiguračního souboru *vsftpd.conf*. Server podporuje například IPv6 protokol, anonymní režim nebo SSL šifrovaní, které umožňuje zabezpečení přihlašovacích údajů, tak aby nebyly zasílány jako řetězec. [42]

6.2 HTTP protokol

Protokol HHTP je určen pro komunikaci s webovými servery, jedná se o komunikaci klient-server nejčastěji na portu 80. Pro komunikaci využívá TCP protokol. HTTP protokol slouží k přenosu *.html* či *.xml* souborů, jedná se o tzv. hypertextové odkazy. Samotná komunikace klient-server je založena na přenosu dotazů (request) a odpovědí (response). [43]

Mezi nejznámější dotazy patří například *GET* pro získání souboru ze serveru například HTML soubor. Dalším dotazem pak může být *HEAD*, tento dotaz je podobný jako ten předchozí, odlišuje se tím, že nepřenáší data, slouží tak například pro získání informací či ověření dostupnosti daného objektu. Dotaz *POTS* umožňuje nahrání souboru na server, popřípadě odeslání dat z formulářů. Dalším dotazem může být *OPTIONS*, který se dotazuje serveru na dostupné metody, *TRACE* pro sledování jednotlivých dotazů. Mezi další dotazy pak patří dotaz *CONNECT, PUT* či *DELETE*. [43]

U HTTP protokolu není zajištěna jeho bezpečnost a zabezpečení tak poskytuje až protokol HTTPS. Tento protokol nedokáže uchovávat svůj aktuální stav, z tohoto důvodu je možné využít tzv. cookies, jedná se o textové soubory, do kterých si server ukládá informace, tyto textové soubory si server ukládá u klienta. Po opětovném spuštění webových stránek si tyto textové soubory server zašle zpět.

Základem HTTP je URL sloužící k identifikaci či specifikaci zdroje uvnitř serveru, zároveň slouží k samotnému přístupu ke zdroji. [43], [44]

Nginx server

Webový server Nginx byl již částečně popsán v kapitole 3. Cílem tohoto serveru je rychlé obsloužení tisíců klientů (C10K problém) s minimálními požadavky na hardware, zejména pak na nízkou spotřebu operační paměti. To je zajištěno možností rozložení zátěže (*load balancing*). Nginx je možné použít i jako reverzní proxy, kdy server přijímá žádosti od klientů, a následně je rozděluje mezi jednotlivé servery, popřípadě poskytuje klientům odpovědi na jejich požadavky. Další možností využití tohoto serveru je tzv. mail proxy server s využitím protokolů SMTP, IMAP či POP3. [45], [46]

Apache server

Apache je webový server s otevřeným zdrojovým kódem, který vznikl kolem roku 1996. Tento server se stal velmi oblíbeným, v poslední době má však velkou konkurenci v podobě Nginx serveru. Apache však má oproti Nginx rozdílnou architekturu, není tak zaměřen na rychlost, ale na spolehlivost, stabilitu a přizpůsobení se systému díky modulům tzv. MultiProcessing (MPM). [45], [47]

Lighttpd server

Webový server Lighttpd, jehož první zmínka se datuje kolem roku 2003, pracuje na podobném principu jako Nginx server, jeho cílem je tak snížení nároků na paměť RAM a zároveň co nejrychleji obsloužit největší počet klientů. Lighttpd se stejně jako Nginx server zabývá problémem C10K. Výhodou oproti Apache serveru může být jeho jednoduchá konfigurace. Ligttpd server obsahuje protokoly *CGI*, kdy tyto protokoly umožňují doinstalování nejrůznějších externích aplikací jako například PHP či Python. I když servery Lighttpd a Nginx jsou velmi podobné jejich hlavní rozdíl je, že Lighttpd funguje pouze jako jeden proces s jedním vláknem, zatímco Nginx funguje jako hlavní proces (master). Tento server je využíván například společnostmi YouTube a Wikipedia. [47], [48], [49]

7 REALIZACE SCÉNÁŘE

Na základě minulých kapitol byl zvolen Ansible jako nejvhodnější nástroj pro automatizaci virtuální infrastruktury. V následujících kapitolách bude popsána prostřednictvím Ansible příprava scénáře pro server, na který bude nainstalován nástroj pro kontejnerovou virtualizaci Docker. Dále budou prostřednictvím tohoto nástroje vytvořeny image a kontejnery s jednotlivými službami a klienty. Všechny tyto kontejnery budou propojeny pomocí sítě, která bude taktéž vytvořena prostřednictvím nástroje Docker. Pro jednotlivé služby bude zajištěno jejich monitorování. V případě jednotlivých kontejnerů zde bude implementováno řešení pro jejich automatické odstranění. Samotná realizace serveru bude rozdělena do následujících částí:

- Hostitelský server
- Vytvoření obrazu disků (image)
- Správa kontejnerů

7.1 Příprava hostitelského serveru

Jak již bylo zmíněno v předcházejících kapitolách pro zajištění samotné komunikace s klientským strojem, kde je nainstalován nástroj Ansible je nutné nainstalovat dostupné aktualizace a balíčky, v tomto případě balíček s programovacím jazykem Python.

7.1.1 Instalace a konfigurace prostředí Docker

Pro tuto realizaci byl vytvořen scénář `docker.yml`, který je umístěn v adresáři `/etc/ansible/books` na klientském stroji. Tento scénář je zobrazen jako Soubor 6.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: instalace balicku
      apt:
        pkg:
        - apt-transport-https
        - ca-certificates
        - curl
        - gnupg-agent
        - software-properties-common
        - python-pip

    - name: GPG key
```

```
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present

    - name: nastaveni – stable repository
      apt_repository:
        repo: deb https://download.docker.com/linux/ubuntu disco
stable
        state: present

    - name: instalace Docker-ce
      apt: update_cache=yes name=docker-ce state=latest

    - name: instalace Docker-ce-cli
      apt: update_cache=yes name=docker-ce-cli state=latest

    - name: instalace containerd.io
      apt: update_cache=yes name=containerd.io state=latest

    - name: Docker modul pro Python
      pip:
        name: docker

    - name: Uninstall python-backports
      pip:
       name: backports.ssl-match-hostname
       state: absent

    - name: Install python-backports
      apt: update_cache=yes name=python-backports.ssl-match-hostname
state=latest

    - name: instalace tshark
      apt: update_cache=yes name=tshark state=latest

    - name: psutil modul pro Python
      pip:
        name: psutil
```

Soubor 6 Instalace a konfigurace prostředí Docker

Při spuštění playbooku je nutné nejdříve stáhnout balíčky, které zajistí, že balíkovací systém *apt* umožní stahování prostřednictvím HTTPS, dále je nainstalován balíček s balíkovacím systémem *python-pip* programovacího jazyka Python. Dále je stažen a přidán oficiální GPG klíč společnosti Docker. Následně jsou stažen a rozbalen samotný Docker. Aby bylo možné Docker ovládat vzdáleně prostřednictvím nástroje Ansible je nutné doinstalovat modul Docker prostřednictvím balíkovacího systému *pip* programovacího jazyku Python. V rámci tohoto modulu bylo nainstalováno několik dalších dílčích balíčků, kde nastal problém s verzí balíčku *python-backports*, kdy tento balíček bylo nutné odinstalovat a poté jej znovu nainstalovat prostřednictvím balíkovacího systému *apt*. Posledními

kroky je pak instalace síťového analyzátoru Tshark a modulu *psutil* pro Python, který umožňuje získat informace o běžících procesech a využití systému v podobě zatížení procesoru, pamětí RAM, disku či sítě. Tento modul a Tshark budou použity pro realizaci částí měření v následujících kapitolách.

7.1.2 Vytvoření virtuální sítě

Po instalaci a následném spuštění nástroje Docker je nutné vytvořit síť, ke které se budou později jednotlivé kontejnery připojovat. Ve výchozím stavu je možné použít již vytvořenou síť, která je v režimu *bridge*, to znamená, že jednotlivé kontejnery včetně hosta získají vlastní IP adresu. Dále je možné pro připojení využít režim *host*, kdy kontejner sdílí všechna síťová rozhraní s hostitelem. V případě režimu *none* jsou veškerá síťová připojení zakázána. Další režimy pak mohou být například *overlay* pro spojení několika Docker démonů či režim *macvlan*, který umožní přidělit kontejnerům vlastní MAC adresu.

V rámci realizace však bude použita síť v režimu *bridge*. Bylo by tak možné využít již předdefinovanou síť, ale u této sítě není možné později prostřednictvím nástroje Ansible přiřadit kontejnerům jejich jednotlivé IP adresy manuálně. U této sítě jsou kontejnerům jejich adresy přidělovány postupně podle jejich spuštění. Z tohoto důvodu byl prostřednictvím nástroje Ansible vytvořen konfigurační soubor `network.yml`, který je zobrazen jako Soubor 7.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: vytvoreni site
      docker_network:
        name: network_bridge
        driver: bridge
        ipam_options:
          subnet: 172.16.0.0/24
          gateway: 172.16.0.1
```

Soubor 7 Vytvoření sítě *network_bridge*

Po spuštění konfiguračního souboru je vytvořena síť s názvem *network_bridge*, která je v režimu *bridge* s adresou podsítě 172.16.0.0/24 a přístupovým bodem 172.16.0.1.

7.2 Vytvoření image

V rámci této kapitoly bude popsán postup vytváření obrazů disků jednotlivých služeb a klientů. Bude zde vytvořeno celkem pět obrazů disků, a to image s FTP, Nginx, Apache a Lighhtpd serverem, poslední pátý image bude pro klienty. K této

konfiguraci bude využit nástroj Ansible, který zajistí nakopírování a stažení potřebných souborů a systémů na hostitelský server. Jednotlivé obrazy disků pak budou vytvářeny prostřednictvím konfiguračních souborů Dockerfile. Veškeré konfigurační soubory budou vytvořeny a umístěny na klientském stroji v adresáři `/etc/ansible/books`. Struktura tohoto adresáře pro vytváření obrazů jednotlivých klientů a služeb je zobrazena jako Soubor 8.

```
books
├── images.yml
└── docker
    ├── apache
    │   ├── Dockerfile
    │   ├── file
    │   └── apache2.conf
    ├── ftp
    │   ├── Dockerfile
    │   ├── file
    │   └── vsftpd.conf
    ├── klient
    │   └── Dockerfile
    ├── lighttpd
    │   ├── Dockerfile
    │   ├── file
    │   └── lighttpd.conf
    └── nginx
        ├── Dockerfile
        ├── file
        └── default
```

Soubor 8 Struktura adresáře *books* pro vytvoření imagů klientů a služeb

Adresář *books* tedy obsahuje playbook *images.yml*, který zajišťuje vytvoření adresářů a nakopírování potřebných souborů do hostitelského serveru. Uvnitř adresáře *books* je pak dále vytvořen adresář *docker*, který obsahuje dílčí podadresáře, jejichž obsahem jsou konfigurační soubory *Dockerfile* pro vytvoření konkrétních imagů, soubor *file*, dále pak obsahují konfigurační soubory dílčích služeb (Apache, FTP, Nginx, Lighttpd). Soubor *file* bude rozebrán později v dalších kapitolách.

Následující část této kapitoly bude věnována jednotlivým konfiguračním souborům nástrojů Ansible a Docker. Konfigurační soubor *images.yml* pro nástroj Ansible je zobrazen jako Soubor 9.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: vytvoreni slozky pro ftp-Dockerfile
      file:
        path: /home/ansible-klient/docker/ftp
        state: directory
```

```
- name: nakopirovani ftp-Dockerfile
  copy: src=/etc/ansible/books/docker/ftp/Dockerfile
        dest=/home/ansible-klient/docker/ftp/Dockerfile

- name: nakopirovani ftp-vsftpd.conf
  copy: src=/etc/ansible/books/docker/ftp/vsftpd.conf
        dest=/home/ansible-klient/docker/ftp/vsftpd.conf

- name: nakopirovani file
  copy: src=/etc/ansible/books/docker/ftp/file
        dest=/home/ansible-klient/docker/ftp/file

- name: ftp kontejner
  docker_image:
    name: ubuntu_ftp
    path: docker/ftp
    state: build

- name: vytvoreni slozky pro klient-Dockerfile
  file:
    path: /home/ansible-klient/docker/klient
    state: directory

- name: nakopirovani Dockerfile
  copy: src=/etc/ansible/books/docker/klient/Dockerfile
        dest=/home/ansible-klient/docker/klient/Dockerfile

- name: klient kontejner
  docker_image:
    name: ubuntu_klient
    path: docker/klient
    state: build

- name: vytvoreni slozky pro nginx-Dockerfile
  file:
    path: /home/ansible-klient/docker/nginx
    state: directory

- name: nakopirovani Dockerfile
  copy: src=/etc/ansible/books/docker/nginx/Dockerfile
        dest=/home/ansible-klient/docker/nginx/Dockerfile

- name: nakopirovani default nginx
  copy: src=/etc/ansible/books/docker/nginx/default
        dest=/home/ansible-klient/docker/nginx/default

- name: nakopirovani file
  copy: src=/etc/ansible/books/docker/nginx/file
        dest=/home/ansible-klient/docker/nginx/file

- name: nginx kontejner
  docker_image:
    name: ubuntu_nginx
    path: docker/nginx
```

```
    state: build

- name: vytvoreni slozky pro apache-Dockerfile
  file:
    path: /home/ansible-klient/docker/apache
    state: directory

- name: nakopirovani Dockerfile
  copy: src=/etc/ansible/books/docker/apache/Dockerfile
        dest=/home/ansible-klient/docker/apache/Dockerfile

- name: nakopirovani apach2.conf
  copy: src=/etc/ansible/books/docker/apache/apache2.conf
        dest=/home/ansible-klient/docker/apache/apache2.conf

- name: nakopirovani file
  copy: src=/etc/ansible/books/docker/apache/file
        dest=/home/ansible-klient/docker/apache/file

- name: apache kontejner
  docker_image:
    name: ubuntu_apache
    path: docker/apache
    state: build

- name: vytvoreni slozky pro lighttpd-Dockerfile
  file:
    path: /home/ansible-klient/docker/lighttpd
    state: directory

- name: nakopirovani Dockerfile
  copy: src=/etc/ansible/books/docker/lighttpd/Dockerfile
        dest=/home/ansible-klient/docker/lighttpd/Dockerfile

- name: nakopirovani lighttpd.conf
  copy: src=/etc/ansible/books/docker/lighttpd/lighttpd.conf
        dest=/home/ansible-klient/docker/lighttpd/lighttpd.conf

- name: nakopirovani file
  copy: src=/etc/ansible/books/docker/lighttpd/file
        dest=/home/ansible-klient/docker/lighttpd/file

- name: lighttpd kontejner
  docker_image:
    name: ubuntu_lighttpd
    path: docker/lighttpd
    state: build

- name: odstraneni pomocnych slozek
  file:
    path: /home/ansible-klient/docker
    state: absent
```

Soubor 9 konfigurační soubor *images.yml* nástroje Ansible

První částí konfigurace je vytvoření adresáře *docker*, ve kterém bude dále vytvořen podadresář *ftp*, do kterého bude nakopírována konfigurace FTP serveru *vsftpd.conf*, soubor *file* a *Dockerfile*. Poté bude vytvořen image *ubuntu_ftp* prostřednictvím konfiguračního souboru *Dockerfile*. Následně bude vytvořen podadresář *klient*, do kterého bude nakopírován konfigurační soubor *Dockerfile*, nakonec bude na základě konfiguračního souboru *Dockerfile* vytvořen image *ubuntu_klient*. Dalším krokem je vytvoření podadresáře *nginx*, do kterého se nakopíruje konfigurace Nginx serveru *default*, soubory *file* a *Dockerfile* pomocí kterého se vytvoří image *ubuntu_nginx*. Úplně stejným způsobem a pořadím budou vytvořeny i zbývající image *ubuntu_apache* a *ubuntu_lighttpd*, tak že se nejdříve nakopíruje *Dockerfile*, následně konfigurační soubor konkrétní služby a nakonec soubor *file*. Po dokončení všech konfigurací a instalací je poslední částí tohoto playbooku odstranění pomocného adresáře *docker* a všech jeho podadresářů z hostitelského serveru. Dostupnost všech těchto imagů je možné ověřit na hostitelském serveru prostřednictvím příkazu `sudo docker images`.

Poslední část této kapitoly bude věnována bližší specifikaci konfiguračních souborů Dockerfile jednotlivých služeb. Pokud bychom chtěli tento typ souboru nějak definovat, v žádném případě se nejedná o soubor napsaný v programovacím jazyku nástroje Docker, jedná se spíše o skript či soubor instrukcí, které se mají provést během vytváření daného obrazu disku. V následujících částech kapitoly tak budou popsány jednotlivé postupy, popřípadě kroky při vytváření obrazů.

```
FROM ubuntu:18.04

RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get -y install vsftpd
RUN rm /etc/vsftpd.conf
COPY vsftpd.conf /etc/vsftpd.conf
RUN useradd -mU student && echo "student:student" | chpasswd
COPY file /home/student/file
```

Soubor 10 Konfigurační soubor Dockerfile pro FTP server

Jako Soubor 10 je zobrazen Dockerfile pro FTP server, kde bude použit image se systémem Ubuntu ve verzi 18.04, který bude dále rozšířen. Nejdříve bude daný systém aktualizován, následně bude doinstalován balíček se samotným vsFTPD serverem. V dalším kroku se provede odstranění výchozího konfiguračního souboru *vsftpd.conf*, místo tohoto souboru bude nakopírován stejný soubor, který je nakonfigurován již podle potřeb. Dále se vytvoří nový uživatel *student* s heslem *student*, prostřednictvím kterého se bude klient k tomuto FTP serveru později přihlašovat. Posledním krokem je nakopírování souboru *file*.

```
FROM ubuntu:18.04

RUN apt-get update
RUN apt-get -y  upgrade
RUN apt-get -y install wget
```

Soubor 11 Konfigurační soubor Dockerfile pro klienta

V případě obrazu disku klienta, který je zobrazen jako Soubor 11 se použil stejný základ v podobě image se systémem Ubuntu 18.04, kdy se po instalaci aktualizací nainstaloval balíček *wget*, který bude později použit pro účely měření.

```
FROM ubuntu:18.04

RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get -y install nginx
COPY default /etc/nginx/sites-enabled/default
COPY file /usr/share/nginx/html/file
```

Soubor 12 Konfigurační soubor Dockerfile pro Nginx server

Konfigurace Nginx serveru, která je zobrazena jako Soubor 12, má stejný základ jako předcházející image. V tomto případě je zde nainstalován balíček *nginx*. Dále je provedeno nakopírování konfigurace serveru *default* a souboru *file* do příslušných adresářů.

```
FROM ubuntu:18.04

RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get -y install apache2
COPY file /var/www/html/file
RUN rm /etc/apache2/apache2.conf
COPY apache2.conf /etc/apache2/apache2.conf
```

Soubor 13 Konfigurační soubor Dockerfile pro Apache server

V případě Apache serveru, jehož konfigurace je zobrazena jako Soubor 13 byl využit taktéž image se systémem Ubuntu 18.04, který byl aktualizován a doplněn o balíček *apache2*, nakopírování souboru *file*, odstranění výchozí konfigurace *apache2.conf* a následné nakopírování upravené konfigurace *apache2.conf*. V této kapitole poslední Dockerfile pro Lighttpd server nebude již uveden, jelikož je identický jako v případě Apache serveru, který je zobrazen jako Soubor 13. Rozdílem je, že bude místo balíčku *apache2* instalován balíček *lighttpd*, stejný postup bude i v případě konfigurace, kdy bude nejdříve odstraněna výchozí a následně nakopírována upravená konfigurace, kdy místo adresáře *apache2* se bude vstupovat do adresáře *lighttpd*.

Tímto jsou všechny potřebné obrazy disků připraveny a je možné přejít k samotnému spouštění jednotlivých kontejnerů prostřednictvím nástroje Ansible.

Jako Tabulka 1 je pak zobrazen výpis všech vytvořených obrazů disků z hostitelského serveru včetně jejich velikostí.

Tabulka 1 Výpis vytvořených obrazů disků na hostitelském serveru

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu_lighttpd	latest	a6cde584c7da	9 days ago	169MB
ubuntu_apache	latest	7411597d88c0	9 days ago	211MB
ubuntu_nginx	latest	c96a77beec04	9 days ago	175MB
ubuntu_klient	latest	f433bdc55dba	9 days ago	109MB
ubuntu_ftp	latest	0e16b982594b	9 days ago	122MB
ubuntu	18.04	72300a873c2c	3 weeks ago	64.2MB

Z tabulky je možné vidět, že původní image Ubuntu ve verzi 18.04 měl velikost 64,2 MB, všechny následně vytvořené image, které z tohoto původního vychází mají znatelně vyšší velikost. V případě tabulky Tabulka 2 je pak detailněji popsáno rozložení kapacity u jednotlivých obrazu disků.

Tabulka 2 Rozložení kapacity obrazu disků

Image	Ubuntu:18.04 [MB]	Update [MB]	Služba [MB]	Wget [MB]	file [MB]
ubuntu_lighttpd	64,2	95,3	52,0	0	21
ubuntu_apache	64,2	95,3	99,4	0	21
ubuntu_nginx	64,2	95,3	61,9	0	21
ubuntu_klient	64,2	95,3	0	7	0
ubuntu_ftp	64,2	95,3	6,4	0	21

Jak již bylo zmíněno, všechny image vycházejí z image Ubuntu 18.04. V rámci instalace a konfigurace jednotlivých obrazů disků prostřednictvím Dockerfile byly nejdříve provedeny aktualizace, kdy po jejich dokončení se zvýšila velikost image přibližně na hodnotu 95,3 MB. Následně pak byly instalovány jednotlivé služby. U obrazu disku *ubuntu_klient* nebyly instalovány žádné služby, ale byl zde instalován balíček *wget* o velikosti 7 MB. V případě všech obrazů disků, které zastávají služby byl prostřednictvím souboru Dockerfile nakopírován zkušební soubor *file* o velikosti přibližně 21 MB.

7.3 Správa kontejnerů

V rámci této kapitoly budou popsány způsoby správy kontejnerů. První část této kapitoly bude věnována spouštěním jednotlivých kontejnerů, druhá část kapitoly

bude pojednávat o zastavení všech vytvořených a spuštěných kontejnerů. Poslední část kapitoly bude pak zaměřena na odstranění všech vytvořených kontejnerů.

7.3.1 Spuštění kontejnerů

Na základě vytvořených obrazů disků jednotlivých služeb, které byly popsány v rámci kapitoly 7.2, je možné přejít ke spuštění samotných kontejnerů. Budou vytvořeny celkem čtyři kontejnery, které budou zastávat služby, konkrétně bude vytvořen kontejner s FTP, Nginx, Apache a Lighttpd serverem. Dále se pak budou vytvářet klientské kontejnery pro jednotlivé služby, to znamená, že budou vytvářeny skupiny klientských kontejnerů, kdy každá skupina bude náležet konkrétní službě. Počet vytvořených klientských kontejnerů bude určovat sám uživatel. Jelikož bude spouštění všech kontejnerů na hostitelském serveru probíhat na klientském stroji s využitím nástroje Ansible, je nutné přejít do adresáře `/etc/ansible/books` a zde následně vytvořit příslušné konfigurační soubory (playbooky) a všechny potřebné náležitosti. Struktura adresáře *books* je zobrazena jako Soubor 14.

```
books
├── ftp.yml
├── nginx.yml
├── apache.yml
├── lighttpd.yml
├── klienti.yml
└── vars
    └── vars.yml
```

Soubor 14 Struktura adresáře *books* pro spuštění kontejnerů

Adresář *books* obsahuje celkem pět konfigurací pro nástroj Ansbile, dále pak adresář *vars*, ve kterém se nachází soubor *vars.yml*, který je svázán, popřípadě propojen s konfigurací klientů konkrétně se souborem *klienti.yml*. Další části této kapitoly budou věnovány již jednotlivým konfiguračním souborům nástroje Ansible.

```
- name: konfigurace hosta
  hosts: <IP severu>
  sudo: True
  tasks:
    - name: spusteni ftp
      docker_container:
        name: ftp
        image: ubuntu_ftp
        networks:
          - name: network_bridge
            ipv4_address: "172.16.0.2"
        detach: yes
        tty: yes
        state: started
```

```
    - name: spusteni sluzby ftp_kontejneru
      command: docker exec -d ftp service vsftpd start
```

Soubor 15 Spuštění kontejneru s FTP serverem

Konfigurační soubor pro spuštění kontejneru s FTP serverem je zobrazen jako Soubor 15. V prvním kroku je vytvořen kontejner s názvem *ftp* na základě image *ubuntu_ftp*. Kontejner je připojen k vytvořené síti *network_bridge*, která byla vytvořena v rámci kapitoly 7.1.2 s IP adresou 172.16.0.2. Pro kontejner je povolen režim *detach*, který zajišťuje běh kontejneru na pozadí a režim *tty*, jedná se o režim *pseudo-TTY* nástroje Docker, který zajišťuje přístup k terminálu. Po vytvoření a spuštění kontejneru *ftp* je posledním krokem provedení příkazu uvnitř tohoto spuštěného kontejneru běžícím na pozadí, kdy příkaz provede spuštění služby vsFTPD.

```
- name: konfigurace hosta
  hosts: <IP severu>
  sudo: True
  tasks:
    - name: spusteni nginx_kontejenru
      docker_container:
        name: nginx
        image: ubuntu_nginx
        networks:
          - name: network_bridge
            ipv4_address: "172.16.0.3"
        detach: yes
        tty: yes
        state: started

    - name: spusteni sluzby nginx_kontejneru
      command: docker exec -d nginx service nginx start
```

Soubor 16 Spuštění kontejneru s Nginx serverem

Playbook pro vytvoření kontejneru s Nginx serverem je zobrazen jako Soubor 16. V první části je vytvořen kontejner *nginx* připojen k síti *network_bridge* s IP adresou 172.16.0.3. Po vytvoření a spuštění na pozadí je uvnitř tohoto kontejneru proveden příkaz pro spuštění služby Nginx. Obdobným způsobem bude vytvořen i kontejner *apache*, který bude připojen k vytvořené síti s IP adresou 172.16.0.4. V rámci tohoto playbooku oproti ostatním konfiguracím pro spuštění kontejnerů se službami bylo nutné použít pro spuštění modulu dva příkazové moduly, kdy v rámci prvního modulu byl aplikován příkaz

```
a2enmod ratelimit
```

Tento příkaz zajišťuje aplikaci změn v serveru, v tomto případě se jedná o aplikaci omezení rychlosti stahování z tohoto serveru. V rámci druhého příkazového modulu nástroje Ansible bude realizován příkaz

```
service apache2 start
```

pro spuštění samotného Apache serveru.

Nakonec je vytvořen a spuštěn *lighttpd* kontejner s IP adresu 172.16.0.5. Tímto jsou všechny kontejnery se službami v podobě FTP, Nginx, Apache a Lightppd vytvořeny. Posledním krokem je vytvoření a spuštění libovolného počtů klientských kontejnerů pro jednotlivé služby. Toho bude docíleno použitím dvou konfiguračních souborů, a to *klienti.yml* a souborem *vars.yml*, který je umístěn v adresáři *vars.* Samotný playbook pro vytvoření klientských kontejnerů je zobrazen jako Soubor 17.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  vars_files:
    - vars/vars.yml
  tasks:
    - name: spusteni ftp_klientu
      docker_container:
        name: "klient_ftp{{item}}"
        image: ubuntu_klient
        networks:
          - name: network_bridge
        detach: yes
        tty: yes
       state: started
      with_sequence: count={{ pocet_ftp }}

    - name: spusteni nginx_klientu
      docker_container:
        name: "klient_nginx{{item}}"
        image: ubuntu_klient
        networks:
          - name: network_bridge
        detach: yes
        tty: yes
        state: started
      with_sequence: count={{ pocet_nginx }}

    - name: spusteni apache_klientu
      docker_container:
        name: "klient_apache{{item}}"
        image: ubuntu_klient
        networks:
          - name: network_bridge
        detach: yes
        tty: yes
        state: started
```

```
    with_sequence: count={{ pocet_apache }}

- name: spusteni lighttpd_klientu
  docker_container:
    name: "klient_lighttpd{{item}}"
    image: ubuntu_klient
    networks:
      - name: network_bridge
    detach: yes
    tty: yes
    state: started
  with_sequence: count={{ pocet_lighttpd }}
```

Soubor 17 Vytvoření a spuštění klientských kontejnerů

Po spuštění konfigurace je nejdříve inicializován adresář se souborem *vars.yml*, dále jsou vykonány jednotlivé části konfiguračního souboru, kde prvním krokem je vytvoření libovolného počtu klientských kontejnerů pro FTP server. Tyto kontejnery budou pojmenovány jako *klient_ftp{{item}}*, kde položka *item* značí pořadí, to znamená, že vytvořené kontejnery budou mít stejný počáteční název, doplněný o pořadové číslo. Všechny tyto vytvořené kontejnery budou připojeny k vytvořené síti *network_bridge*. Rozdílem oproti kontejnerům se službami bude to, že klientským kontejnerům nebude přidělena konkrétní IP adresa, při spuštění tak získají nebližší volnou. Klientské kontejnery stejně jako kontejnery se službami poběží na pozadí, což je zajištěno prostřednictvím příkazů *detach* a *tty*. Poslední částí je pak příkaz *with_sequence*, který zajišťuje vytvoření libovolného počtu kontejnerů. Počet kontejnerů, který je nutné vytvořit je zadán prostřednictvím příkazu *count* a proměnné *počet_ftp*, která je umístěna v souboru *vars.yml*. Stejným způsobem jsou pak spouštěny i zbývající skupiny klientských souborů pro Nginx, Apache a Lighhtpd servery.

```
#Spusteni ftp klientu
pocet_ftp: 0

#Spusteni nginx_klientu
pocet_nginx: 20

#Spusteni apache_klientu
pocet_apache: 20

#Spusteni lighttpd_klientu
pocet_lighttpd: 0
```

Soubor 18 Konfigurační soubor *vars.yml* definující počet klientských kontejnerů

Jako Soubor 18 je zobrazen konfigurační soubor *vars.yml*, kde uživatel může definovat, kolik jednotlivých klientských kontejnerů má být vytvořeno, v tomto případě je vytvořeno dvacet nginx a dvacet apache klientů, v případě ftp a lightttpd

klientů bude vytvořeno celkem nula těchto kontejnerů. Soubor je možné otevřít prostřednictvím libovolného textového editoru s oprávněním *sudo*.

V případě vytvoření kontejnerů celý proces funguje tak, že jsou nejdříve spuštěny kontejnery s jednotlivými službami prostřednictvím nástroje Ansible, následně uživatel v souboru *vars.yml* specifikuje počet jednotlivých klientských skupin. Po zadání a uložení změn je možné spustit konfigurační soubor (playbook) *klienti.yml* prostřednictvím nástroje Ansible.

V tabulce Tabulka 3 je provedeno shrnutí všech spuštěných kontejnerů včetně jejich množství a přidělených IP adres. V tabulce je tak zároveň uvedeno i využití vytvořené virtuální sítě *network_bridge* prostřednictvím nástroje Docker. IP adresy jsou postupně přidělovány od hostitelského serveru po jednotlivé služby. Klientským kontejnerům budou přidělovány IP adresy 172.16.0.6 a vyšší v případě spuštění všech kontejnerů se službami.

Tabulka 3 Seznam spuštěných kontejnerů včetně IP adres

	Počet	**IP**
Hostitelský server	1	172.16.0.1
ftp	1	172.16.0.2
nginx	1	172.16.0.3
apache	1	172.16.0.4
ligttpd	1	172.16.0.5
klient_ftp	n	-
klient_nginx	n	-
klient_apache	n	-
klient_lighttpd	n	-

7.3.2 Zastavení a odstranění spuštěných kontejnerů

Nyní zde bude popsán způsob, kterým bude možné provést zastavení všech běžících kontejnerů ať už dílčích služeb či libovolného množství klientských kontejnerů. K realizaci tohoto postupu bude použit nástroj Ansible, proto je nutné přejít opět do pracovního adresáře tohoto nástroje `/etc/ansible/books`. V tomto adresáři bude vytvořen nový konfigurační soubor *stop.yml*, který je zobrazen jako Soubor 19.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  vars_files:
    - vars/vars.yml
  tasks:
    - name: zastaveni ftp
      docker_container:
        name: ftp
```

```
      state: stopped
    ignore_errors: yes

- name: zastaveni nginx
  docker_container:
    name: nginx
    state: stopped
  ignore_errors: yes

- name: zastaveni apache
  docker_container:
    name: apache
    state: stopped
  ignore_errors: yes

- name: zastaveni lighttpd
  docker_container:
    name: lighttpd
    state: stopped
  ignore_errors: yes

- name: zastaveni ftp_klientu
  docker_container:
    name: "klient_ftp{{item}}"
    state: stopped
  with_sequence: count={{ pocet_ftp }}

- name: zastaveni nginx_klientu
  docker_container:
    name: "klient_nginx{{item}}"
    state: stopped
  with_sequence: count={{ pocet_nginx }}

- name: zastaveni apache_klientu
  docker_container:
    name: "klient_apache{{item}}"
    state: stopped
  with_sequence: count={{ pocet_apache }}

- name: zastaveni lighttpd_klientu
  docker_container:
    name: "klient_lighttpd{{item}}"
    state: stopped
  with_sequence: count={{ pocet_lighttpd }}
```

Soubor 19 Konfigurační soubor *stop.yml* pro zastavení spuštěných kontejnerů

V případě konfiguračního souboru *stop.yml* bude docházet k postupnému zastavování kontejnerů se službami, kdy v tomto případě jsou dílčí úkony doplněny o příkaz *ignore_erros*, který je nastaven na hodnotu *yes*. Tento příkaz je zde použit z důvodu, kdy nastane případ, že některý z těchto kontejnerů nebude spuštěn. V případě, že je spuštěn tento playbook pro zastavení všech kontejnerů a některý z těchto kontejnerů nebude vytvořen či spuštěn, nástroj Ansible o tom informuje

klienta prostřednictvím chyby. To znamená, že playbook je zastaven a zbylé kontejnery již nejsou zastaveny. Tímto je zajištěno, že chyba bude ignorována, a uživatel je tak pouze upozorněn.

Poté budou postupně zastaveny jednotlivé klientské kontejnery. V tomto případě, kdy počet klientských kontejnerů je libovolný, byl opět využit konfigurační soubor *vars.yml*, který je umístěn v podadresáři *vars* v pracovním adresáři nástroje Ansible a v playbooku byl následně přidán příkaz *with_sequence*, kdy množství jednotlivých kontejnerů je uložen v proměnných konfiguračního souboru *vars.yml*.

Po aplikaci scénáře pro zastavení všech spuštěných kontejnerů bude ještě vytvořen konfigurační soubor (playbook) pro odstranění všech těchto zastavených a vytvořených kontejnerů. V pracovním adresáři *books* nástroje Ansible bude vytvořen nový konfigurační soubor *rm.yml*, který je téměř identický jako v případě konfigurace pro zastavení všech kontejnerů. Soubor *rm.yml* opět spolupracuje s konfigurací *vars.yml* v případě klientských kontejnerů. Rozdíl je oproti konfiguraci *stop.yml* v položce *state*, kdy místo stavu *stopped* bude použit stav *absent*, který zajistí odstranění jednotlivých kontejnerů.

7.3.3 Zatížení systému

V rámci této kapitoly bude popsán způsob zatížení kontejnerů s jednotlivými službami. Zátěž bude realizována prostřednictvím klientských kontejnerů, které budou k jednotlivým kontejnerům se službami přistupovat. Na základě předchozí kapitoly 7.3.3 může uživatel pro jednotlivé služby vytvořit libovolný počet klientů. Zatížení bude probíhat tak, že klienti budou přistupovat k jednotlivým službám. V případě kontejnerů se službami v podobě FTP, Nginx, Apache a Lighhtpd serveru je v tomto kontejneru umístěn soubor *file*. Původní myšlenkou bylo realizovat zátěž prostřednictvím konfiguračního souboru *klienti.yml* pro spuštění klientských kontejnerů, kdy při spuštění kontejnerů by byl zároveň spuštěn příkaz pro stažení souboru ze serveru a klientský kontejner by byl následně automaticky ukončen. Tuto možnost nebylo možné realizovat, protože nástroj Ansible jednotlivé kontejnery spouští postupně, to znamená, že je vytvořen kontejner se všemi náležitostmi včetně provedení příkazu pro potřebné stažení souboru z určitého serveru, následně se přejde k vytváření následujícího kontejneru. Z tohoto důvodu není možné jednotlivé servery zatížit, protože soubory jsou stahovány postupně a kvůli tomu se zvyšujícím se počtem klientských kontejnerů se zvyšuje čas provedení dané konfigurace, v tomto případě provedení playbooku *klienti.yml*. Z tohoto důvodu je nutné klientské kontejnery spustit na pozadí stejně jako v případě klientů se službami, dále je nutné vytvořit nový konfigurační soubor *zatez.yml*, prostřednictvím kterého se spustí daná zátěž. K vytvoření konfiguračního souboru bude nutné na straně klienta opět přejít do pracovního adresáře nástroje

Ansible `/etc/ansible/books`. Konfigurační soubor pro spuštění zátěže na klientských kontejnerech je zobrazen jako Soubor 20.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  vars_files:
    - vars/vars.yml
  tasks:
    - name: spusteni ftp_zateze
      command: sudo docker exec -dt klient_ftp{{item}} wget
http://172.16.0.2/file
      loop: "{{range(1, pocet_ftp + 1)|list}}"

    - name: spusteni nginx_zateze
      command: sudo docker exec -dt klient_nginx{{item}} wget
http://172.16.0.3/file
      loop: "{{range(1, pocet_nginx + 1)|list}}"

    - name: spusteni apache_zateze
      command: sudo docker exec -dt klient_apache{{item}} wget
http://172.16.0.4/file
      loop: "{{range(1, pocet_apache + 1)|list}}"

    - name: spusteni lighttpd_zateze
      command: sudo docker exec -dt klient_lighttpd{{item}} wget
http://172.16.0.5/file
      loop: "{{range(1, pocet_lighttpd + 1)|list}}"
```

Soubor 20 Konfigurační soubor *zatez.yml* pro zatížení kontejnerů se službami

V konfiguračním souboru *zatez.yml* prvním krokem byla inicializace konfiguračního souboru *vars.yml*, který udává počet jednotlivých klientských kontejnerů. Následně jsou zde vytvořeny celkem čtyři stejné úkoly, popřípadě operace, kdy každá z nich je určena pro jednotlivou službu, a to FTP, Nginx, Apache či Lighttpd. Úkolem tohoto playbooku je spustit příkaz na pozadí pomocí příkazu nástroje Docker. To znamená, že nástroj Ansible prostřednictvím modulu *command* v jednotlivých kontejnerech spustí příkaz *docker exec -dt*, který umožní spustit příkaz na pozadí uvnitř kontejneru. Další část příkazu určuje jméno kontejneru, ve kterém daný příkaz bude spuštěn, nakonec je uveden příkaz, který se má uvnitř kontejneru provést, v těchto případech se jedná o příkaz *wget* a zadání příslušného odkazu na stažení souboru *file*. V případě jmen kontejnerů je zde uvedena položka *{{item}}*, pomocí které je za názvem kontejneru uvedeno pořadové číslo, které spolu se jménem tvoří celý název jednotlivých kontejnerů. Aby bylo možné příkaz aplikovat pro všechny vytvořené a spuštěné klientské kontejnery, je zde využita smyčka *loop*, která určuje kolikrát bude příkaz proveden. Uvnitř smyčky se pak nachází rozsah hodnot, který se například u FTP klienta pohybuje od *1* do proměnné *pocet_ftp +1*, kdy proměnná *počet_ftp* se nachází v již zmíněném konfiguračním

souboru *vars.yml*. Dále je nutné k této proměnné přičíst hodnotu jedna, tak aby byl příkaz aplikován na všechny kontejnery včetně posledního kontejneru.

Oproti původní myšlence je docíleno toho, že právě díky příkazu nástroje Docker *docer exec -dt*, je zajištěno, že příkaz je spuštěn a dále se pak již nečeká na jeho provedení. I když se na dokončení příkazu nečeká, nastává i zde tak krátká prodleva mezi spuštěním prvního a posledního příkazu. Není možné tak docílit spuštění příkazů ve všech klientských kontejnerech pro danou službu současně.

Na závěr této kapitoly zde bude popsán způsob přípravy souboru *file*, který je nakopírován do jednotlivých kontejnerů zastupující služby prostřednictvím nástroje Ansible. Soubor *file* byl vytvořen v pracovním adresáři nástroje Ansible, v jednotlivých podadresářích adresáře *docker* prostřednictvím příkazu

```
sudo touch file
```

následně byl tento soubor rozšířen prostřednictvím příkazu

```
sudo truncate -s 20M file
```

V tomto případě bude soubor *file* rozšířen na velikost 20 MB.

7.3.4 Aplikace pro měření

Cílem této kapitoly bude popis realizace aplikací pro měření. Budou celkem vytvořeny dvě aplikace, kdy první aplikace bude realizovat měření síťového provozu a je vytvořena prostřednictvím konfiguračních souborů nástroje Ansible. Měření bude probíhat tak, že je zadána doba měření následně po spuštění playbooku bude na hostitelském serveru spuštěno zachytávání paketů prostřednictvím síťového analyzátoru Tshark. V případě ukončení měření je spuštěn druhý playbook, který odešle výsledky na klientský stroj. V případě druhé aplikace, která provádí měření zatížení hostitelského systému uživatel nejdříve zadá požadovaný čas měření, následně spustí playbook, který spustí na hostitelském serveru dílčí aplikace vytvořené v programovacím jazyku Python, kdy je prostřednictvím těchto aplikací měřeno zatížení procesoru, paměti RAM, využití disku včetně jeho zápisových a čtecích rychlostí, nakonec je měřeno vytížení jednotlivých kontejnerů vytvořených v rámci nástroje Docker. Po dokončení měření je možné spustit další playbook, který následně odešle všechny výsledky klientovi. V případě, že je nutné měření z nějakého důvodu přerušit, je možné spustit playbook, který dané měření přeruší a není tak nutné čekat na dokončení samotného měření. V dalších částech této kapitoly zde budou podrobněji rozebrány a popsány vytvořené aplikace pro realizaci měření na hostitelském serveru.

Před samotným popisem jednotlivých aplikací zde bude jako Soubor 21 zobrazena struktura pracovního adresáře *books* nástroje Ansible pro realizaci aplikací zajišťující měření požadovaných parametrů hostitelského serveru a jednotlivých kontejnerů.

```
books
├── tshark_start.yml
├── tshark_stop.yml
├── tshark_send.yml
├── measure_send.yml
├── measure_start.yml
├── measure_stop.yml
├── measure
│   ├── cpu.py
│   ├── disk.py
│   ├── docker.py
│   └── network.py
└── vars
    └── vars.py
```

Soubor 21 Struktura adresáře *books* pro realizaci měření

Konfigurační soubory *tshark_start.yml* a *tshark_send.yml* slouží pro spuštění, a následné odeslání výsledků klientovi v rámci monitorování síťového provozu prostřednictvím síťového analyzátoru Tshark. Klient si může získané výsledky v podobě souboru ve formátu *.pcap* zobrazit například prostřednictvím síťového analyzátoru Wireshark. Pro okamžité přerušení měření je možné použít playbook *tshark_stop.yml*. S playbookem *tshark_start.yml* je pak spojen adresář *vars* a soubor *vars.py* prostřednictvím kterého se zadává doba měření. Zbývající soubory a adresář *measure* v adresáři *books* jsou již věnovány realizaci druhé aplikace, která jak již bylo zmíněno zajišťuje měření zatížení procesoru, paměti RAM a dalších parametrů. I v tomto případě je pro zadání doby měření pro playbook *measure_start.yml* použit soubor *vars.py* v adresáři *vars*. První tři konfigurační soubory (playbooky) slouží pro odeslání výsledků měření klientovi, spuštění měření, či okamžitému přerušení měření. Dále je zde vytvořen adresář *measure*, kdy v adresáři jsou uloženy jednotlivé aplikace pro měření parametrů hostitelského serveru napsané v programovacím jazyku Python.

Monitorování síťového provozu

V případě monitorování sítě je nutné mít na hostitelském serveru nainstalován síťový analyzátor Tshark. Následně v pracovním adresáři *books* nástroje Ansible na klientském stroji je vytvořen konfigurační soubor *tshark_start.yml*, který je zobrazen jako Soubor 22.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  vars_files:
    - vars/vars.py
  tasks:
    - name: vytvoreni tshark adresare
```

```
      file:
        path: /home/ansible-klient/tshark
        state: directory

    - name: vytvoreni souboru
      file:
        path: /home/ansible-klient/tshark/output.pcap
        state: touch
        mode: '0777'

    - name: start capture
      command: /usr/bin/tshark -i <Interface> -w /home/ansible-
klient/tshark/output.pcap
      async: "{{cas}}"
      poll: 0
```

Soubor 22 Konfigurační soubor *tshark_start.yml* pro monitorování síť. provozu

V rámci konfiguračního souboru *tshark_start.yml* je nejdříve inicializován soubor *vars.py*, který je uložen v podadresáři *vars*, následně jsou provedeny jednotlivé úkoly, kdy prvním úkolem je vytvoření adresáře *tshark* na hostitelském serveru, do kterého je v případě druhého kroku vytvořen soubor *output.pcap* s potřebným oprávněním. Posledním krokem této konfigurace je pak spuštění samotného příkazu pro zachytávání paketů prostřednictvím nástroje Tshark. V případě tohoto modulu jsou zde pak využity příkazy *async* a *pool*. Příkaz *async* zajišťuje dobu běhu samotného procesu na hostitelském stroji, v tomto případě nástroje Tshark, který je dán proměnnou *cas*, která je uložena v souboru *vars.py*. Druhým příkazem *pool* je zajištěno, aby nástroj Ansible dále nečekal na dokončení příkazu a pokračoval tak dále v provádění dané konfigurace. Soubor *vars.py* je pak zobrazen jako Soubor 23.

```
#Delka mereni v [s]
cas: 60
```

Soubor 23 Konfigurační soubor *vars.py* pro zadání délky měření

Soubor *vars.py* je pak možné otevřít prostřednictvím textového editoru, kde může uživatel následně zadat dobu měření v sekundách prostřednictvím proměnné *cas*.

Dalším konfiguračním souborem, který je vytvořen je soubor *tshark_stop.yml*, který slouží pro okamžité zastavení měření například z důvodu nějaké chyby. Tento konfigurační soubor využívá modul pro zadávání příkazů, prostřednictvím kterého je aplikován příkaz *pkill* pro okamžité ukončení procesů, v tomto případě procesu *tshark*. Konfigurační soubor *tshark_stop.yml* je zobrazen jako Soubor 24.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
```

```
  tasks:
    - name: stop capture
      command: pkill tshark
```

Soubor 24 Konfigurační soubor *tshark_stop.yml* pro zastavení procesu *tshark*

Posledním konfiguračním souborem pro realizaci tohoto monitorování je soubor *tshark_send.yml*, který je zobrazen jako Soubor 25.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: odeslani souboru klientovi
      fetch:
        src: /home/ansible-klient/tshark/output.pcap
        dest: /home/ubuntu-klient2/output.pcap

    - name: odstaraneni tshark adresare
      file:
        path: /home/ansible-klient/tshark
        state: absent
```

Soubor 25 Konfigurační soubor *tshark_send.yml* pro odeslání výsledků klientovi

Tento konfigurační soubor slouží pro odeslání výsledků měření, v tomto případě odeslání souboru *output.pcap* z hostitelského serveru do klientského stroje prostřednictvím modulu *fetch*. Následně je z hostitelského serveru odstraněn adresář *tshark*.

Monitorování zatížení hostitelského systému

Princip monitorování zatížení hostitelského serveru vychází z monitorování síťového provozu, kdy v rámci pracovního adresáře *books* budou vytvořeny celkem tři plabooky pro spuštění, odeslání či zastavení daného měření, dále bude i v tomto případě využit již vytvořený soubor *vars.py* umístěn v adresáři *vars*, který je zobrazen jako Soubor 23. Následně byl v pracovním adresáři *books* vytvořen podadresář *measure*, uvnitř kterého jsou prostřednictvím programovacího jazyku Python vytvořené dílčí skripty pro realizaci jednotlivých měřeních. Playbook pro spuštění měření je zobrazen jako Soubor 26.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  vars_files:
    - vars/vars.py
  tasks:
    - name: vytvoreni adresare pro mereni
      file:
        path: /home/ansible-klient/measure
```

```
    state: directory

- name: nakopirovani cpu_ram_measure
  copy: src=/etc/ansible/books/measure/cpu.py
        dest=/home/ansible-klient/measure/cpu.py

- name: nakopirovani disk_measure
  copy: src=/etc/ansible/books/measure/disk.py
        dest=/home/ansible-klient/measure/disk.py

- name: nakopirovani network_measure
  copy: src=/etc/ansible/books/measure/network.py
        dest=/home/ansible-klient/measure/network.py

- name: nakopirovani vars.py
  copy: src=/etc/ansible/books/vars/vars.py
        dest=/home/ansible-klient/measure/vars.py

- name: uprava vars.py
  replace:
    path: /home/ansible-klient/measure/vars.py
    regexp: ':'
    replace: '='

- name: nakopirovani docker_measure
  copy: src=/etc/ansible/books/measure/docker.py
        dest=/home/ansible-klient/measure/docker.py

- name: start cpu_ram_measure
  command: python /home/ansible-klient/measure/cpu.py
  async: "{{cas}}"
  poll: 0

- name: start disk_measure
  command: python /home/ansible-klient/measure/disk.py
  async: "{{cas}}"
  poll: 0

- name: start network_measure
  command: python /home/ansible-klient/measure/network.py
  async: "{{cas}}"
  poll: 0

- name: start docker_measure
  command: python /home/ansible-klient/measure/docker.py
  async: "{{cas}}"
  poll: 0
```

Soubor 26 Konfigurační soubor *measure_start.yml pro spuštění měření*

V rámci souboru *measure_start.yml* je tak nejdříve inicializován soubor *vars.py*, následně je vytvořen pracovní adresář *measure* na hostitelském serveru, do kterého jsou dále nakopírované jednotlivé skripty a soubor *vars.py* z adresáře *vars*. Tento soubor slouží pro zadání délky měření jak pro jednotlivé playbooky zajišťující

spuštění měření, tak pro jednotlivé skripty, z tohoto důvodu bylo nutné tento soubor nakopírovat i na hostitelský server a vytvořit jej jako soubor ve formátu *.py*. Nástroj Ansible umožňuje použít i tento jiný typ souboru oproti výchozímu, který je ve formátu *.yml* při dodržení syntaxe nástroje Ansible, to znamená, že proměnná musí být vytvořena ve formátu název proměnné, dvojtečka a její hodnota. Oproti tomu zápis proměnné pro programovací jazyk Python je ve formátu název proměnné, rovná se a její hodnota. Z tohoto důvodu je zde dalším krokem úprava nakopírovaného souboru *vars.py*, kdy prostřednictvím modulu *replace* a příkazu *regexp* je vyhledána v tomto souboru dvojtečka a následně je příkazem *replace* nahrazena znakem rovná se. Poslední úkoly konfigurace jsou pak věnovány spouštěním jednotlivých skriptů prostřednictvím modulu *command* a příkazů *async* a *poll*, které zajistí, že program poběží po dobu stanovenou proměnnou *cas* a zároveň nástroj Ansible nečeká na provedení jednotlivých příkazů.

Následně je vytvořen konfigurační soubor *measure_stop.yml*, který slouží k okamžitému přerušení měření. Playbook *measure_stop.yml* je stejný jako soubor *tshark_stop.yml*, který je zobrazen jako Soubor 24, kdy místo procesu *tshark* je použit proces *python*.

Třetím konfiguračním souborem je pak playbook *measure_send.yml* pro odeslání výsledků měření, který je zobrazen jako Soubor 27.

```
- name: konfigurace hosta
  hosts: <IP serveru>
  sudo: True
  tasks:
    - name: odeslani cpu_ram_measure
      fetch:
        src: /home/ansible-klient/measure/cpu_usage.txt
        dest: /home/ubuntu-klient2/cpu_usage.txt

    - name: odeslani disk_measure
      fetch:
        src: /home/ansible-klient/measure/disk_usage.txt
        dest: /home/ubuntu-klient2/disk_usage.txt

    - name: odeslani network_measure
      fetch:
        src: /home/ansible-klient/measure/network_usage.txt
        dest: /home/ubuntu-klient2/network_usage.txt

    - name: odeslani docker_measure
      fetch:
        src: /home/ansible-klient/measure/docker_usage.txt
        dest: /home/ubuntu-klient2/docker_usage.txt

    - name: odstraneni adresare measure
      file:
        path: /home/ansible-klient/measure
```

```
        state: absent
```

Soubor 27 Konfigurační soubor *measure_send.yml* pro odeslání výsledků měření

V první části konfiguračního souboru jsou odeslány výsledky měření získané z jednotlivých částí programu (skriptů) do klientského stoje v podobě textových souborů se změřenými hodnotami. Nakonec je smazán pracovní adresář *measure* na hostitelském serveru.

Zbývající část kapitoly bude věnována jednotlivým skriptům vytvořených prostřednictvím programovacího jazyku Python. Skript pro měření zatížení procesoru a paměti RAM je zobrazen jako Soubor 28.

```
import psutil
import time
from time import gmtime, strftime
from vars import cas

value=1
file = open("/home/ansible-klient/measure/cpu_usage.txt", "w+")

file.write("%s %s %s %s %s" %('Time','        ','CPU(%)','
','RAM(%)'))
file.write("\n")

for value in range(1,cas):
    file.write("%s %s %4.1f %s %4.1f" % (time.strftime("%H:%M:%S",
gmtime()), '    ',psutil.cpu_percent(interval=1), '    ',
psutil.virtual_memory().percent))
    file.write("\n")
```

Soubor 28 Aplikace *cpu.py* pro měření zatížení procesoru a pamětí RAM

Aplikace využívá knihovny *psutil* umožňující monitorování zařízení, dále knihovnu *time* pro zajištění časové synchronizace s dalšími aplikacemi, popřípadě skripty, následně je zde inicializován soubor *vars.py*. Prvním krokem aplikace je vytvoření textového souboru *cpu_usage.txt*, následně jsou do tohoto souboru zapsány názvy zastupující jednotlivé sloupce, a to *Time*, *CPU(%)*, *RAM(%)*. Nakonec je vytvořen *for* cyklus, který je dán počtem cyklů v rozsahu 1 až hodnotou proměnné *cas*, která je uložena v souboru *vars.py*. V rámci tohoto *for* cyklu jsou zapsány do souboru *cpu_usage.txt* hodnoty funkce *strftime* v podobě času z knihovny *time*, následně pak funkci *cpu_percent* s hodnotou zatížení procesoru udávanou v procentech a nakonec funkci *virtual_memory.percent*, která udává využití pamětí RAM v procentech, obě tyto proměnné jsou z knihovny *psutil*.

Další skript, který je použit při monitorování zatížení hostitelského systému je *disk.py*, který slouží k měření rychlosti čtení a zápisu disku. Skript pro měření využití pevného disku je zobrazen jako Soubor 29.

```
import psutil
import time
from time import gmtime, strftime
from vars import cas

value=1
file = open("/home/ansible-klient/measure/disk_usage.txt", "w+")

for value in range(1,cas):
  vychozi = psutil.disk_io_counters()
  time.sleep(1)
  aktualni = psutil.disk_io_counters()
  cteni = aktualni.read_bytes - vychozi.read_bytes
  zapis = aktualni.write_bytes - vychozi.write_bytes

  file.write("%s %s %4.1f %s %9.0f %s %9.0f" %
(time.strftime("%H:%M:%S", gmtime()), '    ',
psutil.disk_usage('/').percent, '    ', cteni, '     ',zapis))
  file.write("\n")
```

Soubor 29 Aplikace *disk.py* pro měření využití pevného disku

Stejně jako v předcházejícím případě jsou zde využity knihovny *psutil*, *time*, i v tomto případě je zde využit soubor *vars.py*. Prvním krokem aplikace je vytvoření textového souboru *disk_usage.txt*, dále je vytvořen *for* cyklus v rozsahu 1 až hodnota proměnné *cas* souboru *vars.py*. Uvnitř cyklu je pak vytvořena proměnná *vychozi*, jejíž obsahem je obsah funkce *disk_io_counters()*, následně je aplikace na určitou dobu pozastavena, poté je vytvořena proměnná *aktualni* jejíž obsahem je obsah funkce *disk_io_counters()*. Následně jsou vypočteny hodnoty čtení a zápisu disku jako rozdíl proměnných *vychozi* a *aktualni*. Nakonec je proveden zápis do souboru v podobě času, využití disku v procentech a následně jsou zapsány vypočtené hodnoty čtení a zápisu disku.

Poslední skript, který je použit v rámci tohoto měření je *docker.py* zobrazen jako Soubor 30.

```
import os

os.system("sudo docker stats >> /home/ansible-
klient/measure/docker_usage.txt")
```

Soubor 30 Aplikace *docker.py* pro měření zatížení jednotlivých kontejnerů

V rámci tohoto skriptu je využita knihovna *os*, prostřednictvím které je aplikován příkaz *system*, který provede příkaz, v tomto případě je spuštěn výpis zatížení kontejnerů `docker stats`, jehož výstup je následně zapsán do textového souboru *docker_usage.txt*.

7.3.5 Grafické rozhraní

Během této závěrečné práce byly vytvářeny různé konfigurace (playbooky) pro správu ať už samotného hostitelského serveru, tak jednotlivých kontejnerů. V případě předchozích kapitol byly tyto konfigurace popsány jako například konfigurace pro instalaci a konfiguraci nástroje Docker, vytvoření sítě, instalace obrazů disků, či správa a vytvoření kontejnerů. V případě správy hostitelského serveru budou její součástí i konfigurace pro vypnutí či restartování samotného serveru nebo konfigurace pro správu nástroje Docker v podobě výpisu dostupných imagů, kontejnerů či sítí. Celkový obsah pracovního adresáře *books* tak bude poměrně rozsáhlý. V případě samotné konfigurace hostitelského serveru tak uživatel musí znát například cestu k pracovnímu adresáři *books*, dále pak musí znát názvy všech vytvořených konfigurací, popřípadě si vypsat obsah adresáře. To v případě spuštění jedné jediné konfigurace znamená využití hned několika příkazů, popřípadě sdružení několika dílčích příkazů do jednoho dlouhého. Na základě těchto nedostatků byla na klientském stroji vytvořena aplikace, která jednotlivé konfigurace spustí, a ty jsou následně realizovány. Nejedná se o plnohodnotnou aplikaci, která je schopna samostatného běhu, ale spíše o grafické rozhraní, kdy spuštění je provedeno prostřednictvím terminálu. Uživatel pak prostřednictvím tlačítek zvolí požadovaný úkon, který je následně realizován. Grafické rozhraní bylo vytvořeno s využitím programovacího jazyka Python a knihovny Qt5, v následujících částech kapitoly bude popsán postup samotné realizace tohoto grafického rozhraní.

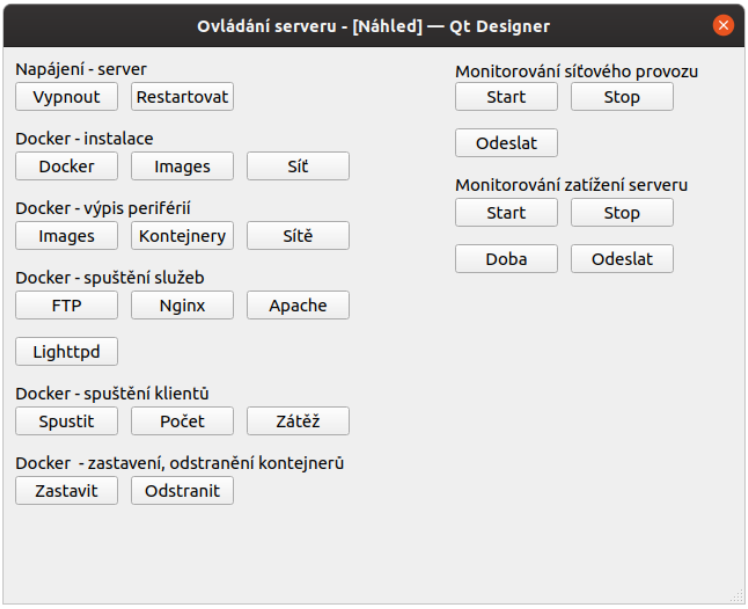
Prvním krokem je vytvoření samotného grafického rozhraní, to bude provedeno prostřednictvím již zmíněné knihovny Qt5, kterou je možné nainstalovat prostřednictvím příkazu

```
sudo apt-get install python-pyqt5
```

Po dokončení instalace této knihovny pak bude nutné ještě nainstalovat software Qt5 Designer, prostřednictvím které bude vytvořena grafická část této aplikace. Nástroj Qt5 Designer je možné nainstalovat prostřednictvím příkazů

```
sudo apt-get install qttools5-dev-tools
sudo apt-get install qttools5-dev
```

Následně je možné spustit samotný nástroj a vytvořit grafické rozhraní pro danou aplikaci. Grafické rozhraní je zobrazeno jako Obrázek 2.



Obrázek 2 Grafické rozhraní pro ovládání hostitelského serveru

V případě takovéto vytvořené aplikace bude soubor uložen ve formátu *.ui*, v tomto případě bude vytvořeno grafické rozhraní s názvem *app.ui*. Dále je nutné do kódu doplnit funkce, které zajistí spuštění jednotlivých úkonů. Předtím je ale nutné vytvořené grafické rozhraní převést z formátu *.ui* do *.py* programovacího jazyka Python, to se provede prostřednictvím terminálu a zadáním následujícího příkazu

```
pyuic5 -x app.ui -o app.py
```

Výsledkem je pak zdrojový kód programovacího jazyka Python, který se dále otevře například v textovém editoru či vývojovém prostředí PyCharm, kde je nutné kód doplnit o funkce provádějící spouštění jednotlivých playbooků z pracovního adresáře *books*. Tyto funkce je pak nutné přiřadit jednotlivým tlačítkům. Funkce je vytvořena následně

```
def pushButton_pressed(self):
    os.system("sudo ansible-playbook /etc/ansible/books/shutdown.yml")
```

V tomto případě se jedná o příklad funkce *pushButton_pressed*, která provede příkaz *os.system* z importované knihovny *os*, kdy prostřednictvím tohoto příkazu je možné spustit systémový příkaz, v tomto případě příkaz pro spuštění konfigurace *shutdown.yml* pro vypnutí hostitelského serveru prostřednictvím nástroje Ansible. Podobným způsobem jsou vytvořeny i funkce pro zbývající tlačítka. Následně je nutné všechny tyto funkce přiřadit jednotlivým tlačítkům. Přiřazení funkce *pushButton_pressed* tlačítku *push_button* je provedeno následně

```
self.pushButton.clicked.connect(self.pushButton_pressed)
```

V aplikaci je implementován i soubor *vars.yml* pro zadání počtu klientů, které se mají následně vytvořit, uživateli po stisknutí tlačítka počet se tento soubor otevře prostřednictvím textového editoru gedit. Funkce pro tlačítko *Počet* bude následující

```
def pushButton_15_pressed(self):
    os.system("sudo gedit /etc/ansible/books/vars/vars.yml")
```

Do aplikace byl taky vložen mechanismus pro realizaci měření v podobě monitorování síťového provozu a zatížení systému. Stejně jako v předcházejících případech se jedná o vytvořená tlačítka, která spouštějí dané konfigurační soubory. V rámci měření je zde podobně jako v případě souboru *vars.yml* pro nastavení počtu klientských kontejnerů vytvořeno tlačítko *Doba*, prostřednictvím kterého je spuštěn editor gedit, ve kterém je otevřen soubor *vars.py*, kde uživatel zadá dobu měření.

Jakmile jsou vytvořeny všechny funkce a přiřazeny jednotlivým tlačítkům, je možné spustit samotnou aplikaci prostřednictvím příkazu

```
python app.py
```

8 REALIZACE MĚŘENÍ

V rámci této kapitoly budou popsány jednotlivé výsledky měření prostřednictvím aplikací pro monitorování síťového provozu a zatížení systému v podobě hostitelského serveru, které byly popsány v kapitole 7.3.4. Samotné měření bude rozděleno do několika částí, kdy prvním krokem bude analýza fyzického hardwaru, na kterém bude měření probíhat, dále pak budou provedena dílčí měření. Jelikož klientský stroj s nástrojem Ansible a hostitelský server jsou realizovány s využitím virtualizace bude nutné přidělit těmto jednotlivým strojům část fyzického hardwaru. Měření tak proběhne ve dvou fázích, kdy v rámci první fáze bude provedeno měření na hostitelském serveru s výchozími parametry fyzického hardware. Ve druhé fázi pak bude hostitelskému systému přidělena větší část fyzického hardwaru. Realizace měření je tak rozdělena do následujících částí:

- Analýza hardwaru
- Měření s výchozími parametry hardwaru
- Měření s upravenými parametry hardwaru
- Vyhodnocení měření

8.1 Analýza hardwaru

Tato kapitola bude věnována analýze fyzického hardwaru, kde budou popsány samotné parametry testovací sestavy spolu s možností přidělení tohoto hardwaru jednotlivým virtuálním strojům. Parametry testovací sestavy jsou následující:

- CPU: AMD Ryzen 2600 6 fyzických jader, 12 vláken, 3,4 GHz
- RAM: 2 x 8 GB DDR4 2666 MHz
- HDD: ADATA Ultimate SU800

V rámci systému s nástrojem Ansible a hostitelského serveru tak bude možné využít část tohoto fyzického hardwaru, jelikož jako hostitelský systém je použit operační systém Windows 10, zmíněný disk je určen pro běh pouze virtuálních strojů, hostitelský operační systém využívá jiný disk. Jako Tabulka 4 je pak zobrazen přehled dostupného hardwaru pro využití v rámci virtualizace.

Tabulka 4 Přehled dostupného hardwaru pro virtuální stroje

	CPU [-]	RAM [MB]
Celkem	12	16 384
Využitelné	6	11 433
Ansible	2	2 048
Server	4	9 385

U celkového využití hardwaru je možné v případě procesoru využít až 12 virtuálních CPU (vláken) a celkem 16 384 MB pamětí RAM. Jak již bylo zmíněno část těchto prostředků využívá i hostitelský operační systém Windows 10. Pro virtualizaci a samotné systémy v podobě operačního systému s nástrojem Ansible a hostitelský server je možné využít celkem 6 virtuálních CPU a 11 433 MB paměti RAM. V rámci celého měření pak operační systém s nástrojem Ansible využíval 2 virtuální CPU a 2 048 MB paměti RAM, to znamená, že pro hostitelský server je možné využít až 4 virtuální CPU a 9 385 MB paměti RAM.

8.2 Měření s výchozími parametry hardwaru

V rámci této kapitoly bude popsáno měření zatížení hostitelského serveru s výchozími parametry nastavení jeho hardwaru. Nastavení těchto parametrů je zobrazeno jako Tabulka 5.

Tabulka 5 Výchozí nastavení hardwaru pro virtuální stroje

	CPU [-]	RAM [KB]	HDD [KB]	
			Celkem	Volných
Ansible	2	2 038 536	20 509 264	11 794 612
Server	1	1 006 532	10 252 564	3 471 396

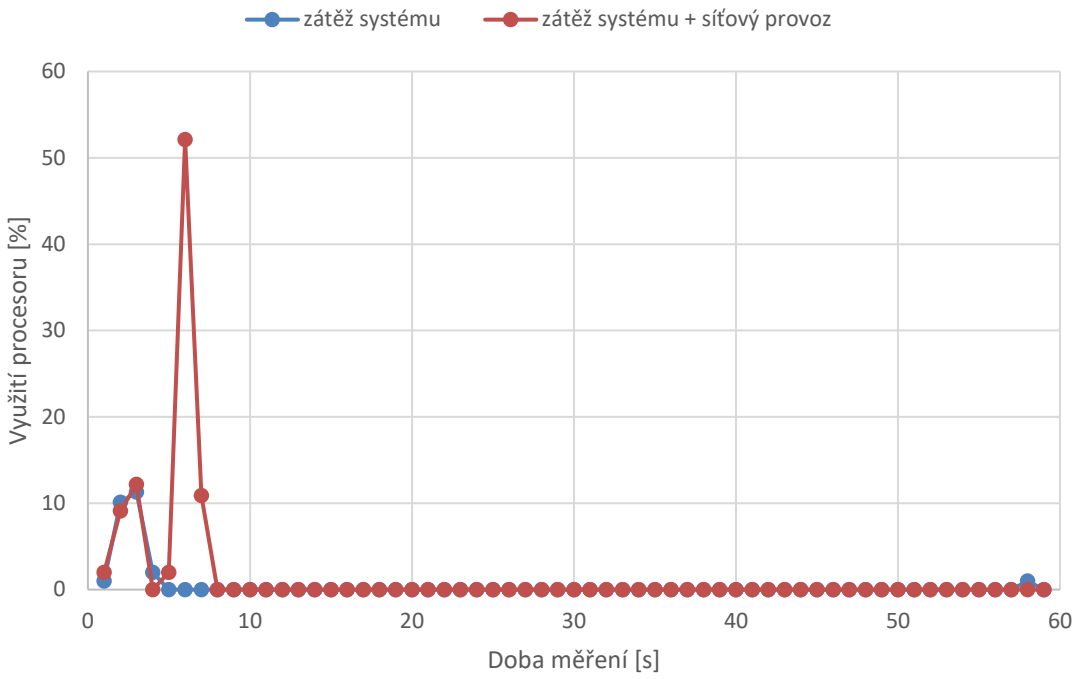
Klientskému operačnímu systému s nástrojem Ansible jsou přidělena dvě jádra procesoru, 2 GB operační paměti a kapacita disku o velikosti přibližně 20 GB. V případě hostitelského serveru je využito jedno jádro procesoru, 1 GB operační paměti a celkem 10 GB je vyhrazeno pro disk. Z celkové kapacity je pak možné využít přibližně 3 GB pro realizaci měření.

Při realizaci měření pak bude nejdříve provedeno tzv. měření naprázdno, kdy bude nejdříve změřeno zatížení samotného spuštěného serveru, následně se bude toto měření opakovat již se spuštěnými aplikacemi pro měření. Tím bude zjištěno, jak dané aplikace zatěžují daný server. Následně pak bude provedeno měření se spuštěnými kontejnery, následně pak bude provedeno měření se zátěží, kdy bude zjištěn počet klientů, které je možné obsloužit.

8.2.1 Měření naprázdno

V případě pouze spuštěného serveru, kdy nejsou spuštěné žádné kontejnery, služby či aplikace pro měření je využití procesoru v rozmezí 0-0,7 %, využití paměti RAM je 36 %. Následně byla provedena dvě měření, kdy v prvním případě bylo spuštěno měření prostřednictvím aplikace pro měření zátěže systému. Následně bylo provedeno druhé měření, kdy byla spuštěna i aplikace pro monitorování síťového provozu. Z výsledků těchto dvou měření pak bude možné určit nároky, popřípadě

zatížení systému těmito dvěma aplikacemi. Výsledky těchto dvou měření budou zobrazena ve dvou společných grafech, kdy první graf zobrazuje vytížení procesoru, druhý graf zobrazuje využití paměti RAM při jednotlivých měřeních. Grafická závislost zobrazující využití procesoru na době měření v čase 59 sekund je zobrazena jako Graf 1.



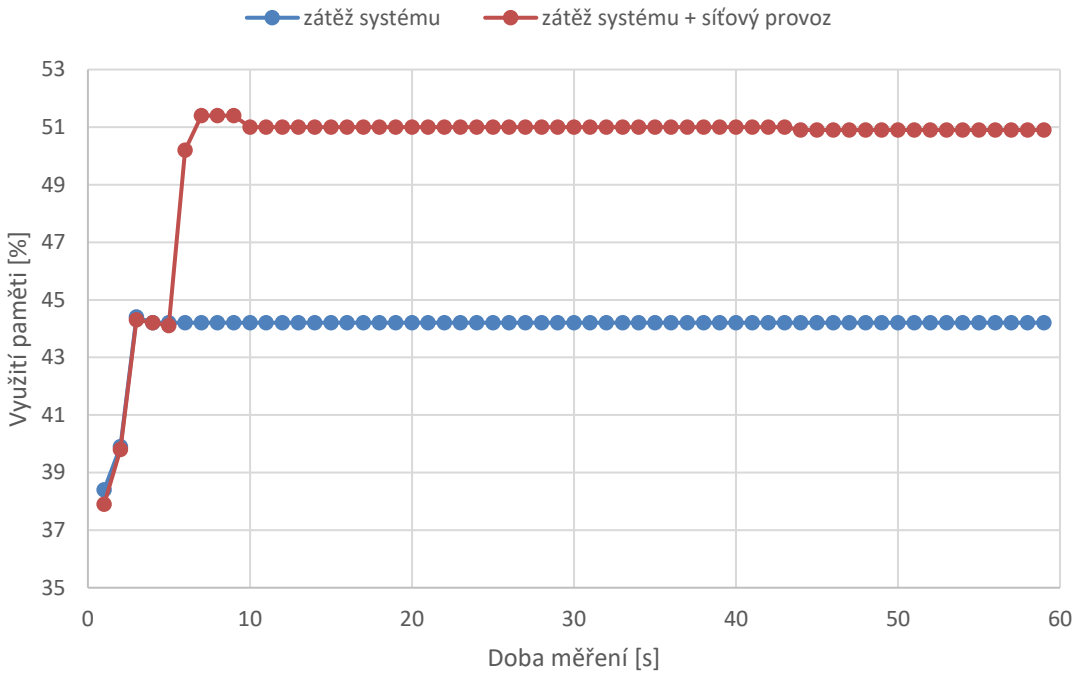
Graf 1 Zatížení procesoru aplikacemi pro realizaci měření

Graf 1 zaznamenává zatížení systému při spouštění dílčích aplikací pro měření v daném čase. V případě grafické závislosti *zátěž systému* je vidět zatížení procesoru v daném čase způsobené spouštěním dílčích aplikací aplikace pro monitorování zatížení hostitelského systému popsané v kapitole 7.3.4. První bod zaznamenává spuštění aplikace pro měření zátěže procesoru a paměti RAM, v rámci druhého bodu je spuštěna aplikace pro měření využití disku a ve třetím bodu je spuštěna aplikace pro měření zatížení jednotlivých kontejnerů. Z této charakteristiky je tak možné vidět, že aplikace pro monitorování zátěže hostitelského systému zatíží procesor do hodnoty 12,2 % v případě jejího spuštění, následně pak aplikace na procesor nemá žádný vliv. Následující charakteristika *zátěž systému + síťový provoz* zaznamenává průběh zatížení procesoru v případě, kdy jsou spuštěny obě aplikace pro realizaci měření, a to aplikace monitorování síťového provozu a monitorování zatížení hostitelského systému. Jedná se téměř o identickou charakteristiku, rozdíl pak nastává v bodech 5-6, kdy dojde ke skokovému zvýšení zátěže až do hodnoty 52,1 %. Tento skok je způsoben

spuštěním aplikace monitorování síťového provozu. Pokles na hodnotu 0, který zaznamenává bod 4 je způsoben dobou, kdy uživatel spustil druhý konfigurační soubor (playbook) s aplikací pro Monitorování síťového provozu.

Grafická závislost zobrazující využití pamětí RAM na době měření v čase 59 sekund je zobrazena jako Graf 2.

Následné nulové využití procesoru je způsobeno tím, že nejsou spuštěny žádné kontejnery a nedochází tak k zatížení procesoru vlivem dílčích aplikací pro realizaci měření.

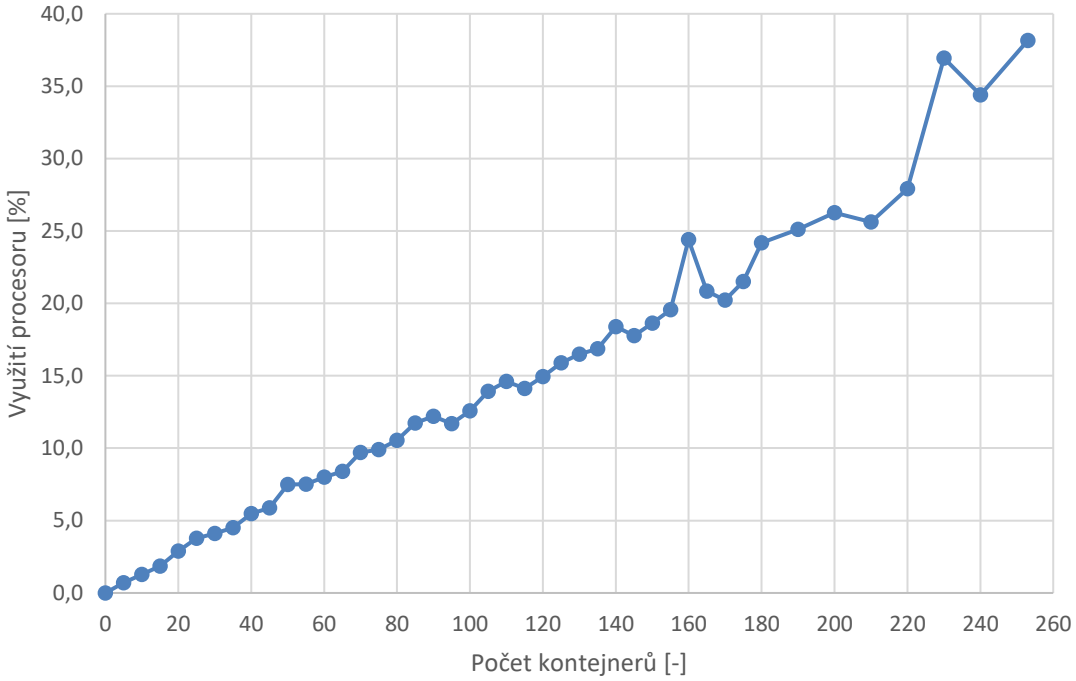


Graf 2 Zatížení paměti aplikacemi pro realizaci měření

Graf 2 zaznamenává zvýšení využití paměti RAM při spuštění aplikací pro realizaci měření. Charakteristika *zátěž systému* zaznamenává zvýšení zátěže paměti v případě spuštění aplikace pro monitorování hostitelského systému, první tři body charakteristiky zaznamenávají spuštění dílčích aplikací pro měření. Využití paměti RAM je v případě pouze spuštěného serveru přibližně 36 %, po spuštění měření se hodnota ustálila na 44,2 %, to znamená v případě spuštění měření nárust přibližně o 8 %. V případě grafické závislosti *zátěž systému + síťový provoz* je opět zátěž zvýšena přibližně o tuto hodnotu, následně v bodu 6 dojde ke zvýšení využití paměti způsobené spuštěním aplikace monitorování síťového provozu, následně je využití pamětí ustáleno na hodnotě 51 %. Z těchto výsledků vyplývá, že aplikace realizující měření zatíží paměť o 15 %. Samotné monitorování síťového provozu pak využívá přibližně 7 % paměti.

8.2.2 Měření se spuštěnými službami a klienty

V rámci této kapitoly bude realizováno měření zatížení hostitelského serveru postupným spouštěním jednotlivých kontejnerů. Kontejnery budou postupně spouštěny, kdy nejdříve dojde ke spuštění kontejnerů se službami, které mají v rámci vytvořené sítě pevně nastavené IP adresy, následně pak budou spouštěny kontejnery klientské. U vytvořené sítě, která byla popsána v kapitole 7.1.2 je možné vytvořit až 253 kontejnerů. V rámci tohoto měření bylo provedeno celkem 44 dílčích měření o délce 59 sekund prostřednictvím aplikace pro monitorování zatížení hostitelského serveru, kdy krok měření byl do hranice 180 kontejnerů nastaven na počet 5, poté byl krok zvýšen na hodnotu 10. Z těchto dílčích měření byly stanoveny průměrné hodnoty zatížení procesoru a paměti RAM. Aby měření nebylo ovlivněno samotnou aplikací, byly průměrné hodnoty zatížení stanoveny z času měření 5-59. Grafická závislost zobrazující vytížení procesoru při zvyšujícím se počtu kontejnerů je zobrazena jako Graf 3.

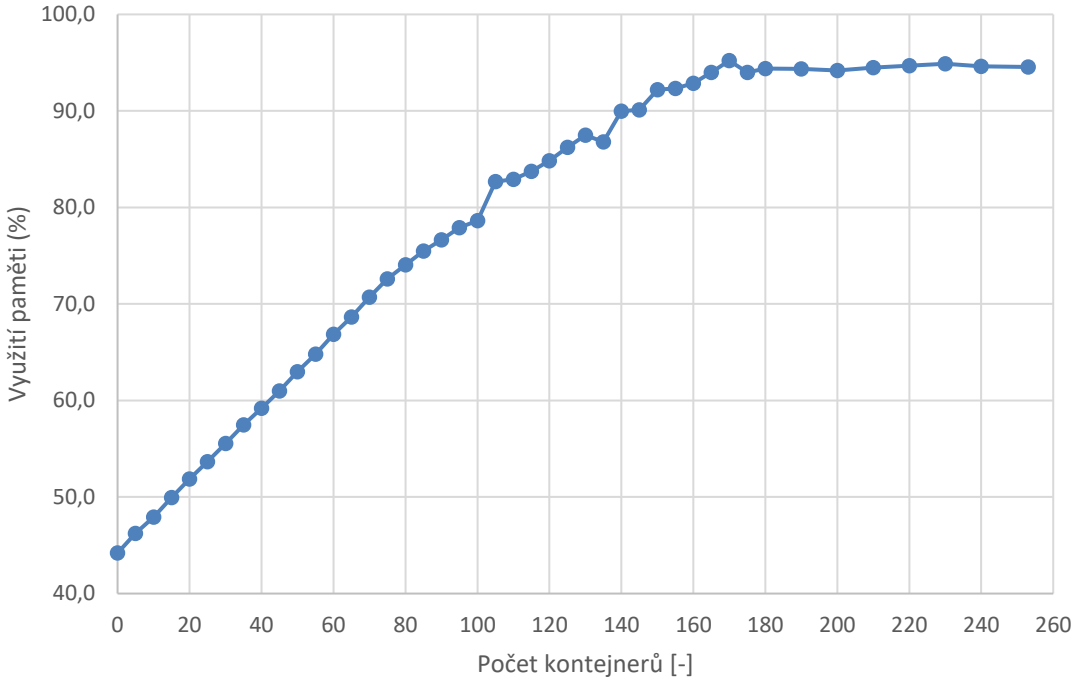


Graf 3 Zatížení procesoru při zvyšujícím se počtu kontejnerů

V případě grafické závislosti Graf 3 je zobrazeno i tzv. nulté měření, kdy nebyly spuštěny kontejnery a zatížení procesoru bylo 0 %. Zatížení procesoru stoupá téměř lineárně. Od hodnoty 160 spuštěných kontejnerů pak dochází k většímu kolísání. Na samotné zatížení procesoru mají kromě zvyšujícího se počtu kontejnerů i dílčí měřící aplikace. V tomto případě se jedná zejména o aplikaci

realizující měření využití kontejnerů, konkrétně statistky *docker stats*, jejíž nároky na hardware rostou se zvyšujícím se počtem kontejnerů.

Grafická závislost zobrazující využití paměti RAM při zvyšujícím se počtu kontejnerů je zobrazena jako Graf 4.



Graf 4 Zatížení paměti při zvyšujícím se počtu kontejnerů

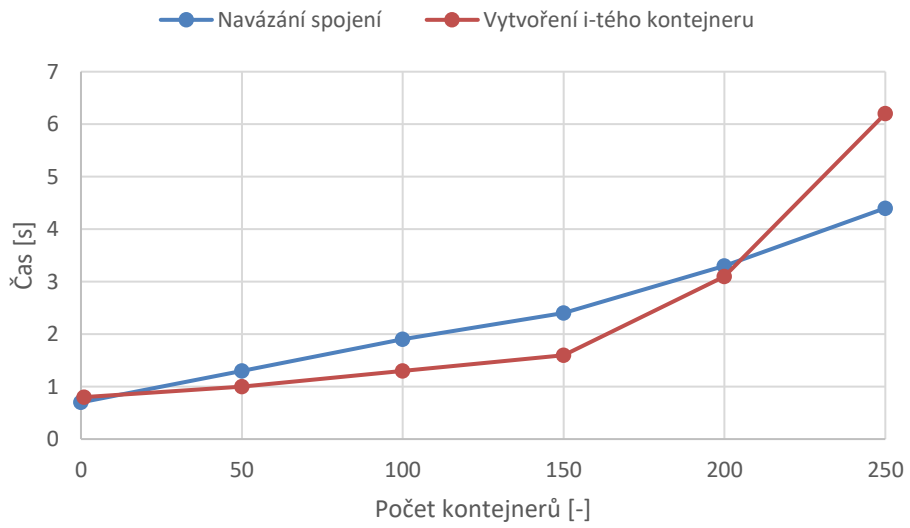
Graf 4 zaznamenává i nulté měření při spuštění pouze aplikace pro měření, kdy využití paměti RAM je 44,2 %, z této hodnoty 8 % je využíváno měřící aplikací. Se zvyšujícím se počtem kontejnerů dochází k lineárnímu nárustu využití paměti RAM do hodnoty 95,2 % při 170 spuštěných kontejnerech. Následně dochází k postupnému ustálení vytížení paměti RAM na hodnotě 94-95 %.

Z obou grafických závislostí je možné vidět, že zatížení jak procesoru, tak paměti se zvyšuje lineárně se zvyšujícím se počtem kontejnerů, v případě většího počtu kontejnerů dochází k postupnému snižování a následnému ustálení hodnot využití paměti. Ze statistik d*ocker stats* je pak možné vidět, že v daný okamžik nejsou některé kontejnery dostupné. Aby nedocházelo k úplnému vyčerpání paměti a ukončování procesů a tím i samotného nástroje Docker se samotný nástroj pokouší tato rizika snížit, tím může dojít k ukončení některých kontejnerů.

Při postupném zatížení hostitelského serveru dochází i k prodloužení doby navázání spojení mezi klientem a hostitelským serverem, kdy například po spuštění playbooku pro vytvoření a spuštění kontejnerů je při spouštění dalších konfigurací (playbooků) vidět zpoždění v rámci navazování spojení klient – server. Graf 5

a charakteristika *Navázání spojení* zobrazuje dobu navazování spojení, při určitém počtu spuštěných kontejnerů, které představují zátěž. V rámci tohoto měření tak byl spuštěn určitý počet kontejnerů, následně pak byl spuštěn nějaký playbook, například pro výpis spuštěných kontejnerů, imagů či virtuálních sítí, kdy byla následně změřena doba navázaní daného spojení.

K určitému zpoždění či prodlevě dochází i v případě samotného playbooku například u již zmiňovaného playbooku pro vytvoření a spuštění určitého počtu kontejnerů. Při tomto postupném spouštění a vytváření kontejnerů se zvyšuje doba, za kterou je daný kontejner vytvořen a spuštěn. Graf 5 a grafická závislost *Vytvoření i-tého kontejneru* zobrazuje dobu, kterou trvá vytvoření a spuštění daného kontejneru prostřednictvím playbooku a nástroje Ansible.



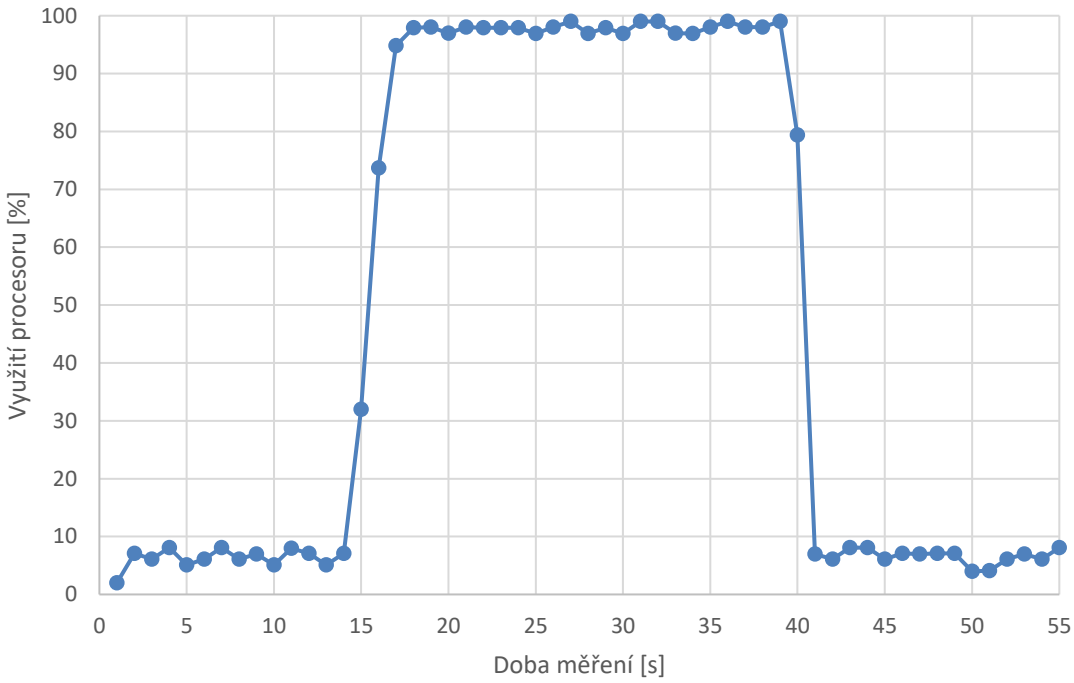
Graf 5 Odezvy při zvyšujícím se počtu kontejnerů

8.2.3 Měření zátěže

Tato kapitola bude věnována měření zátěže hostitelského serveru v podobě vytvořeného zkušebního souboru o velikosti 20 MB, který je umístěn v kontejnerech realizující služby. Klientské kontejnery pak budou k těmto kontejnerům se službami přistupovat a stahovat daný soubor. Cílem této kapitoly tak bude zjistit kolik klientských kontejnerů je hostitelský server schopen zpracovat a jaké má hardwarové nároky. Na základě předchozích kapitol bylo zjištěno, že je možné vytvořit celkem 253 kontejnerů, kdy v případě úplného využití paměti RAM může dojít k ukončování kontejnerů. V případě jednoho kontejneru se službami bylo možné obsloužit celkem 169 klientských kontejnerů, kdy nebyla spuštěna žádná z měřících aplikací, následně bylo měření ukončeno chybou z důvodu nedostatku místa na pevném disku. Počet 169 klientských kontejnerů odpovídá stažení souborů

o celkové velikosti 3 380 MB. V případě spuštěné aplikace pro monitorování hostitelského systému pak bylo obslouženo 167 klientů, to odpovídá stažení souborů o velikosti 3 340 MB. Pokud je spuštěna aplikace pro monitorování síťového provozu je počet obsloužených klientů snížen o polovinu, kdy velikost provozu zachycena nástrojem Tshark odpovídá velikosti stažených souborů.

Aby bylo možné realizovat měření a zachytit průběh využití procesoru a paměti RAM, byl na základě grafických závislostí Graf 3, Graf 4 stanoven pracovní bod, který určuje, kolik klientských kontejnerů bude vytvořeno pro realizaci měření. Bude tak vytvořeno celkem 50 klientských kontejnerů, kdy zátěž procesoru by se měla pohybovat okolo 7,5 % a využití paměti RAM by mělo být přibližně 63 %, dále pak bude vytvořen a spuštěn kontejner realizující služby. Využití procesoru v průběhu měření je zobrazeno jako Graf 6.

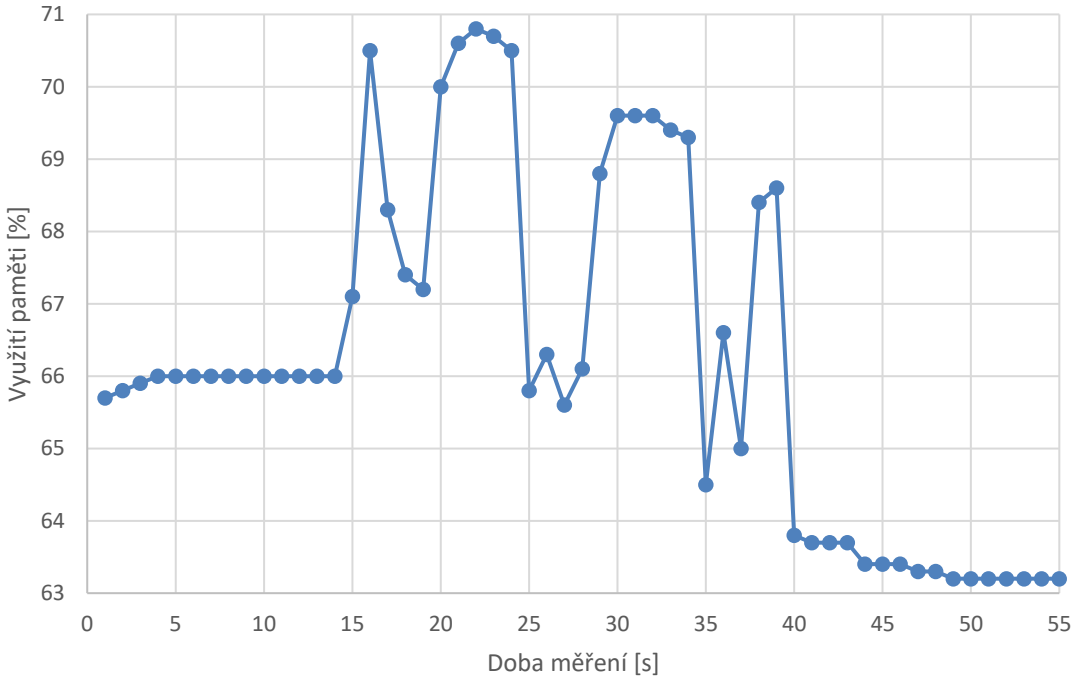


Graf 6 Zatížení procesoru při stahování

Graf 6 zobrazuje, jak je zatížen procesor při spuštění zátěže. V čase měření 1-14 je zobrazeno využití procesoru v případě, kdy je spuštěno celkem 50 klientských kontejnerů a jeden kontejner představující služby, v tomto případě FTP server. Následně dochází ke spuštění zátěže prostřednictvím nástroje Ansible, kdy je nejdříve navázáno spojení klient-server, následně od času měření 17 je spuštěno stahování do času 41, kdy je stahování ukončeno. V případě hodnot v čase 40 a 41 dochází již k poklesu zatížení procesoru, kdy probíhá dokončení stahování v rámci

posledních spuštěných klientských kontejnerů. Následně dochází k zpětnému snížení využití procesoru.

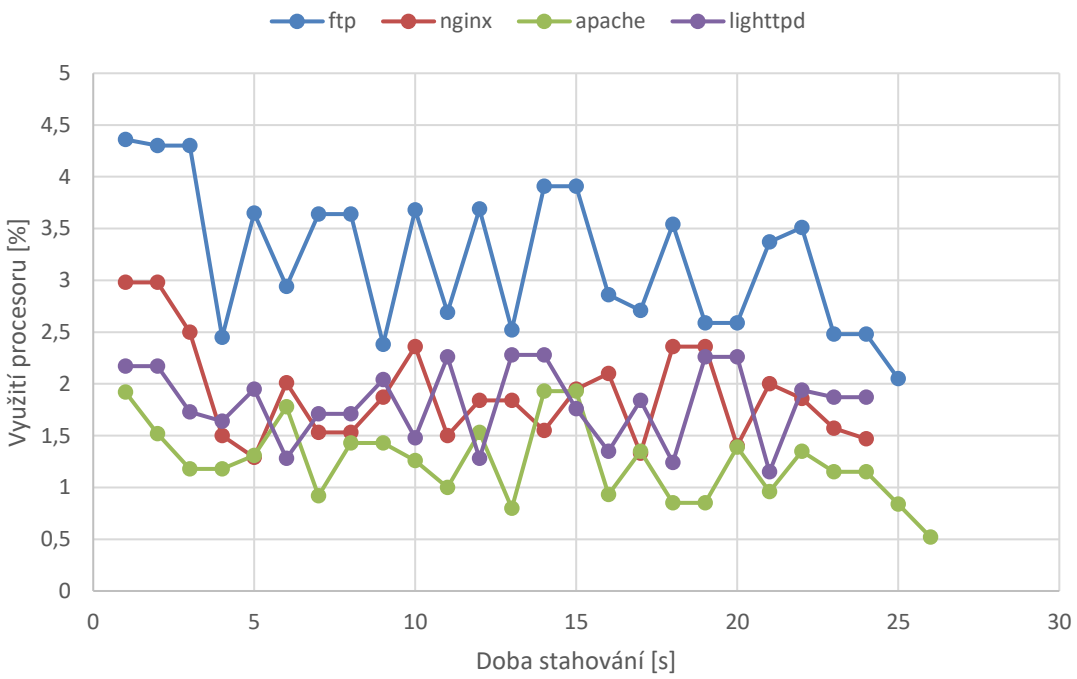
Grafická závislost zobrazující využití paměti RAM v průběhu měření je zobrazena jako Graf 7.



Graf 7 Zatížení paměti při stahování s výchozími parametry HW

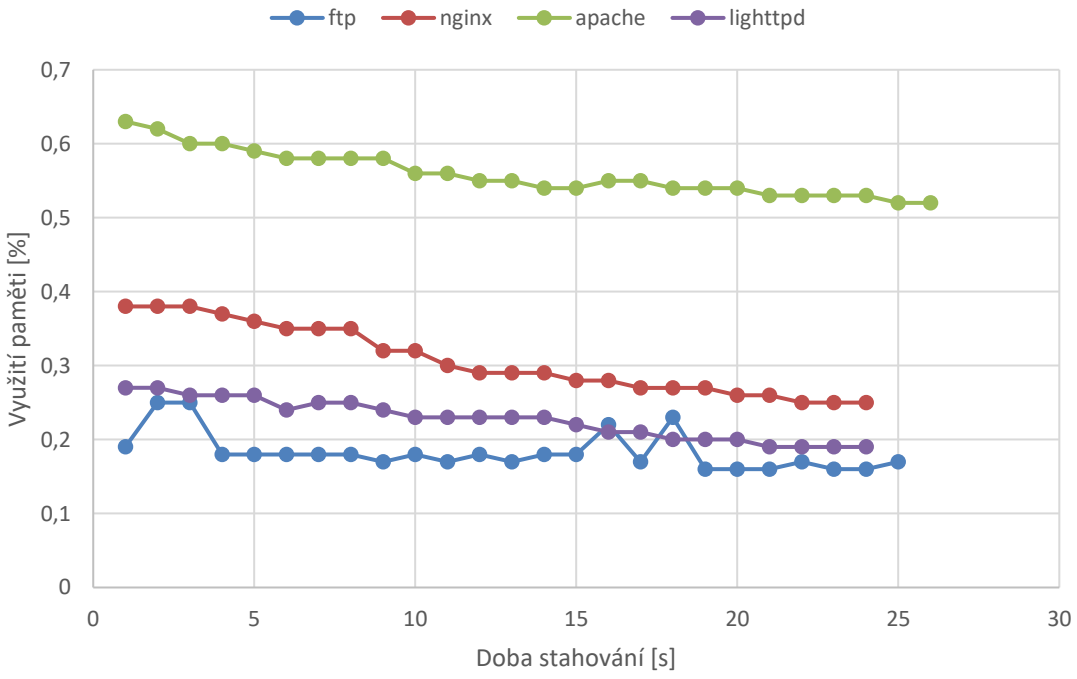
Graf 7 zobrazuje využití paměti RAM v průběhu měření při stahování souboru prostřednictvím klientských kontejnerů. V čase 1-14 jsou spuštěny všechny kontejnery. Následně dochází ke spuštění měření a zvýšení zatížení paměti. Kolísání je způsobeno různým zatížením kontejnerů v průběhu stahování a zejména měřícími aplikacemi, které zaznamenávají probíhající stahování. Po dokončení stahování dochází k poklesu a postupnému ustálení.

Měření, které je zde popsáno prostřednictvím grafických závislostí Graf 5, Graf 6 bylo provedeno pro všechny použité FTP, Ngix, Apache a Lighttpd protokoly. Na základě těchto měření bylo možné prostřednictvím *docker stats* změřit využití procesoru a paměti RAM a spolu s využitím stanovit čas stahování v rámci kontejnerů s těmito službami. Grafická závislost zobrazující využití procesoru v rámci jednotlivých kontejnerů se službami je zobrazena jako Graf 8.



Graf 8 Využití procesoru jednotlivými službami

Graf 8 obrazuje využití procesoru jednotlivými službami, kdy nejvíce je využíván v případě FTP serveru. Nejméně je pak využíván v případě Apache serveru. Závislost zobrazující využití paměti RAM těmito službami je zobrazena jako Graf 9.



Graf 9 Využití paměti jednotlivými službami

Graf 9 zobrazuje využití paměti RAM jednotlivými službami, kdy největší nároky na paměť RAM má Apache server, nejmenší pak FTP server. Jak již bylo zmíněno, na základě Docker statistik a záznamu o využití disku bylo možné určit dobu stahování všech souborů. V případě Lighttpd a Nginx serveru činí doba stahování 24 s, u FTP serveru 25 s, nejdéle stahování trvalo u Apache serveru, a to 26 s.

8.3 Měření s upravenými parametry hardwaru

V rámci tohoto měření bude realizováno měření s již upravenými parametry hostitelského serveru, které jsou zobrazeny jako Tabulka 6.

Tabulka 6 Upravené nastavení hardwaru pro virtuální stroje

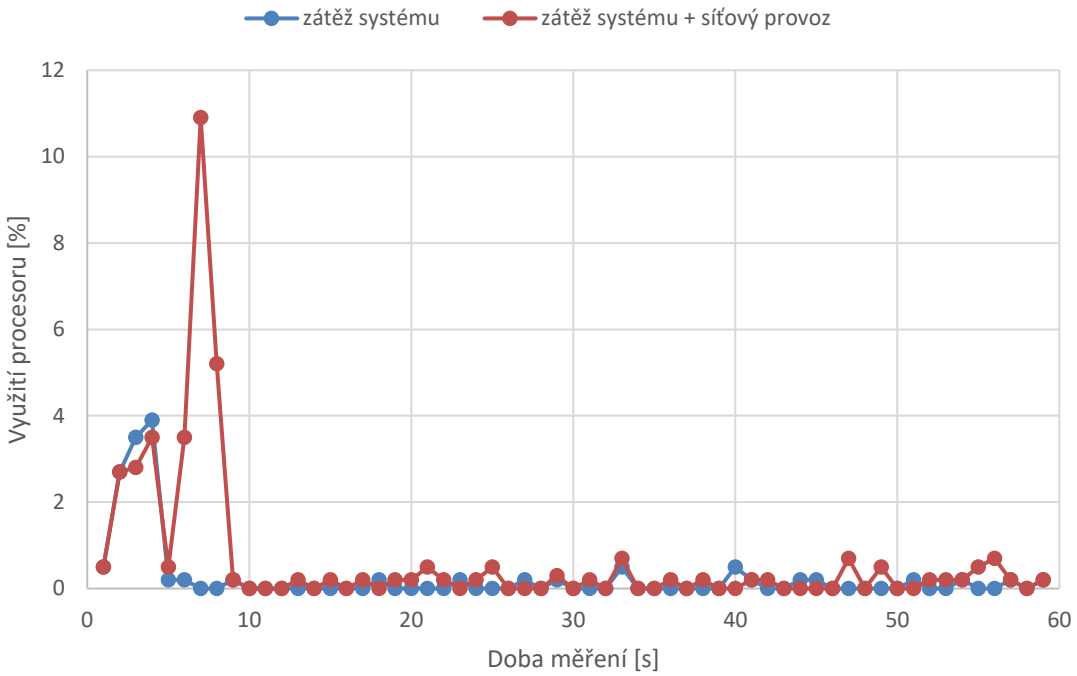
	CPU [-]	RAM [KB]	HDD [KB]	
			Celkem	Volných
Ansible	2	2 038 536	20 509 264	11 794 612
Server	4	9 341 884	51 340 768	40 187 340

Klientskému stroji jsou ponechány stejné parametry, hostitelskému serveru jsou nově přiděleny 4 jádra procesoru, 9 385 MB paměti RAM a 50 GB kapacity pevného disku, k dispozici je pak 40 GB z této kapacity.

Stejně jako v předcházejícím měření popsaném v kapitole 8.2 bude měření realizováno ze tří dílčích částí, a to měření naprázdno, dále bude provedeno měření se spuštěnými kontejnery, a nakonec bude realizováno měření se zátěží.

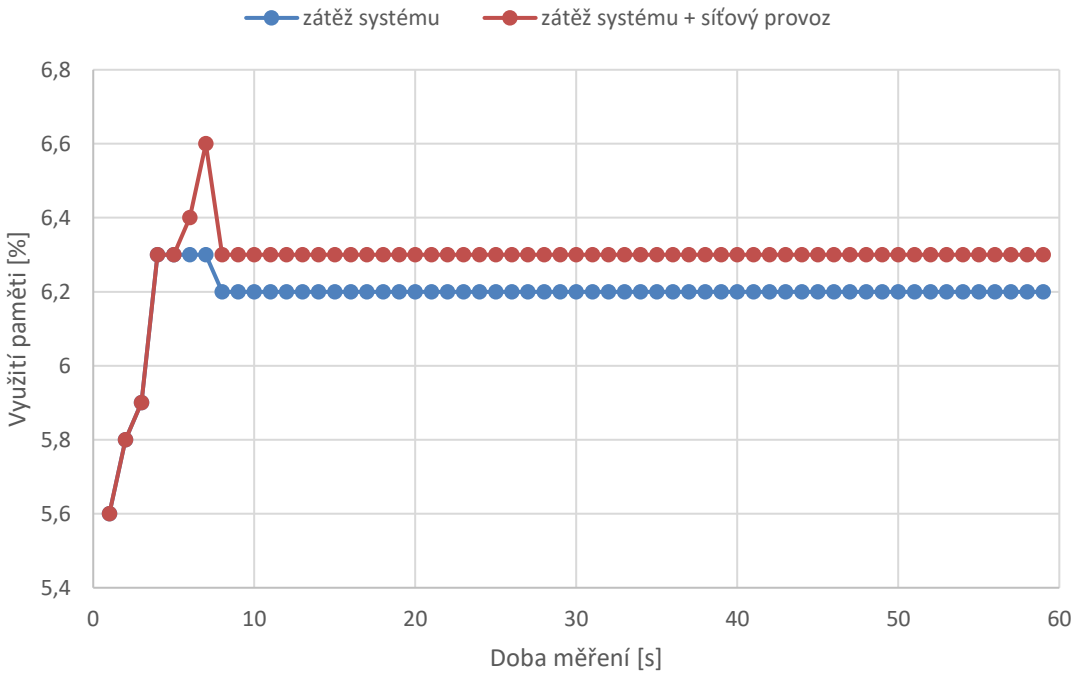
8.3.1 Měření naprázdno

Po upravení parametrů hardwaru hostitelského serveru se zatížení procesoru pohybuje v rozmezí 0-0,2 %, zatížení pamětí je 5,5 %. Následně bylo opět provedeno měření se spuštěnými aplikacemi pro realizaci měření. V prvním kroku bylo provedeno měření prostřednictvím aplikace pro monitorování hostitelského systému, následně bylo provedeno druhé měření, kdy bylo kromě monitorování hostitelského systému spuštěno i monitorování síťového provozu. Obě měření byla realizována v době 59 sekund. Grafická závislost zobrazující zatížení procesoru aplikace pro realizaci měření je zobrazena jako Graf 10.



Graf 10 Zatížení procesoru aplikacemi pro realizaci měření s upraveným HW

Graf 10 zobrazuje identickou grafickou charakteristiku jako Graf 1. Zatížení při startu aplikací jsou přibližně čtyřikrát menší než v případě měření s výchozími parametry. Po ustálení se hodnoty využití procesoru pohybují mezi 0-0,7 %.

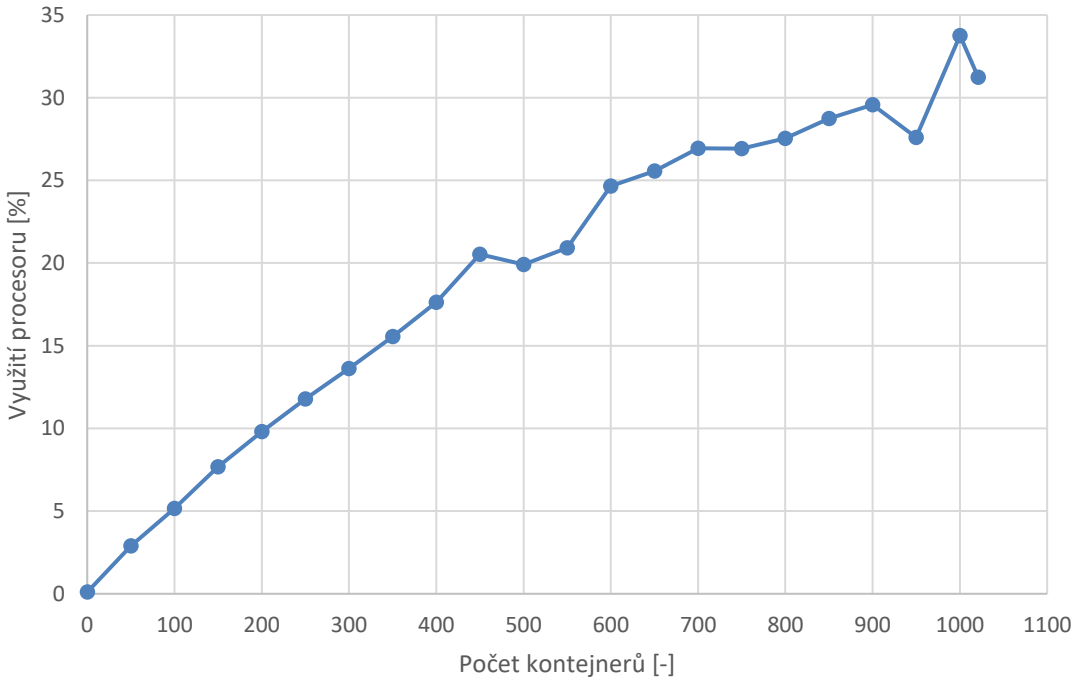


Graf 11 Zatížení paměti aplikacemi pro realizaci měření s upraveným HW

Graf 11 zobrazuje zatížení paměti RAM při spuštění aplikací pro realizaci měření, kdy při spouštění aplikací realizující monitorování hostitelského systému jsou oba průběhy identické, u grafické závislosti *zátěž systému* dochází ke snížení a ustálení využití paměti na hodnotě 6,2 %. V případě grafické závislosti *zátěž systému + síťový provoz* dochází ke skokovému zvýšení zatížení na hodnotu 6,6 %, což je způsobeno spouštěním aplikace pro monitorování síťového provozu, následně dojde k ustálení využití paměti RAM na hodnotě 6,3 %. Aplikace pro monitorování zatížení hostitelského systému zatíží paměť o 0,7 %, aplikace pro monitorování síťového provozu pak o 0,8 %. Zatížení pamětí je nyní téměř osmkrát menší, něž v případě výchozích parametrů hardwaru.

8.3.2 Měření se spuštěnými službami a klienty

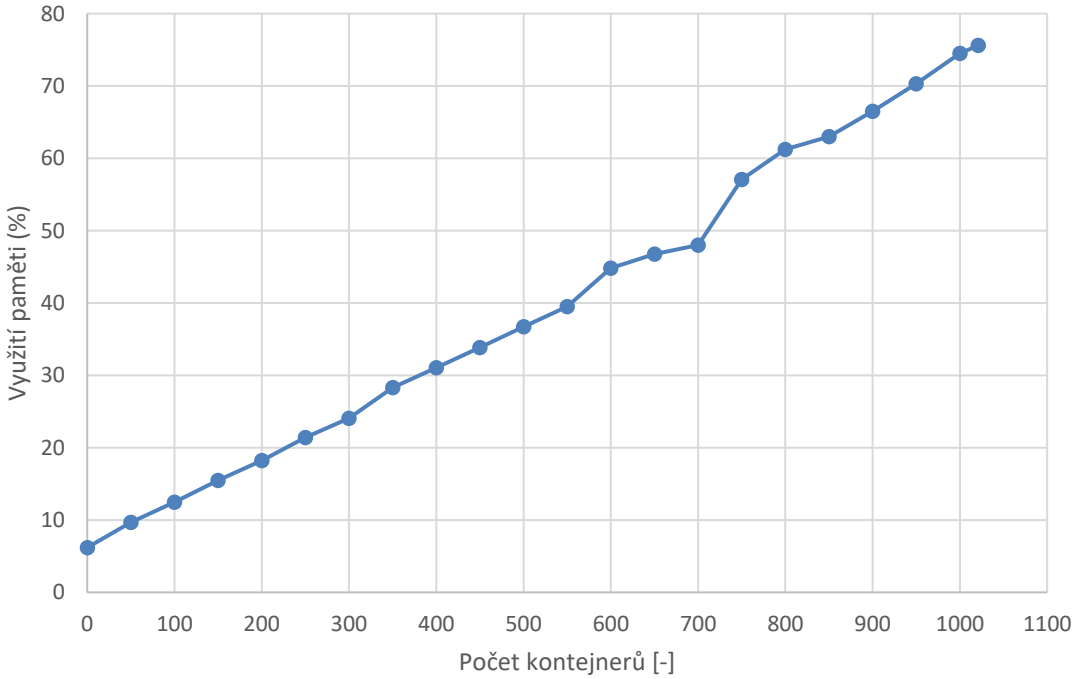
Pro realizaci měření bylo nutné kromě parametrů hardwaru hostitelského serveru upravit i vytvořenou virtuální síť. V rámci výchozích parametrů byla vytvořena virtuální síť prostřednictvím nástroje Docker, s adresou podsítě 172.16.0.0/24, kdy prostřednicím této sítě bylo k dispozici celkem 254 IP adres, z toho 253 adres použitelných v rámci kontejnerů. Aby bylo možné realizovat měření s upravenými parametry hardwaru serveru bylo nutné u virtuální sítě změnit její masku z původcích 24 na 22. Tím bylo docíleno zvětšení adresního rozsahu z původních 254 IP adres na 1022, kdy IP adresa 172.16.0.1 je přidělena hostitelskému serveru, zbývajících 1021 IP adres je možné využít v rámci kontejnerů.



Graf 12 Zatížení procesoru při zvyšujícím se počtu kontejnerů s upraveným HW

Graf 12 zobrazuje zatížení procesoru při postupném zvyšování počtu kontejnerů, na zátěž mají kromě zvyšujícího se počtu kontejnerů vliv i aplikace realizující měření, v tomto případě aplikace pro monitorování zatížení hostitelského serveru. V případě výchozích parametrů bylo při 253 spuštěných kontejnerech naměřeno zatížení procesoru 38,2 %, v případě upravených parametrů hardwaru je pak při stejném počtu spuštěných kontejnerů naměřena zátěž 11,8 %. V rámci upravených parametrů hardwaru a virtuální sítě bylo možné realizovat měření u 1021 spuštěných kontejnerů při zátěži 31,2 %.

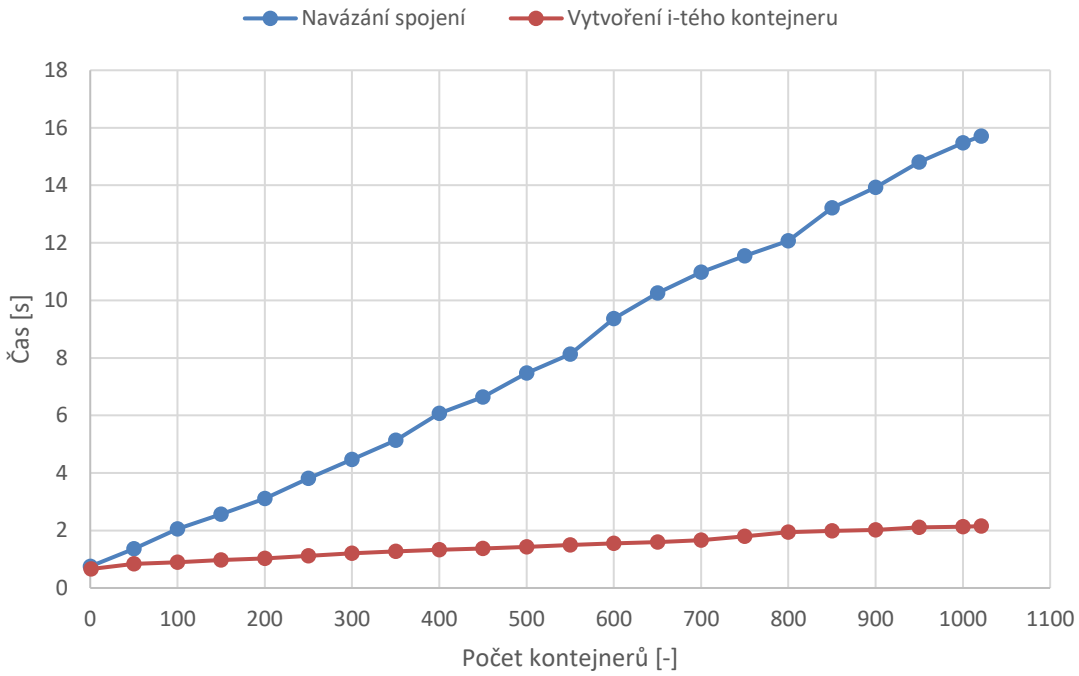
Grafická závislost Graf 13 zobrazuje využití paměti RAM při zvyšujícím se počtu spuštěných kontejnerů.



Graf 13 Zatížení paměti při zvyšujícím se počtu kontejnerů s upraveným HW

Graf 13 zobrazuje lineární průběh využití paměti RAM při postupném zvyšování počtu kontejnerů. V porovnání s grafem Graf 4 s výchozím parametry hardwaru bylo zjištěno, že daný průběh je lineární do hranice 120 spuštěných kontejnerů při zatížení 93,5 %, v rámci tohoto měření bylo při 150 spuštěných kontejnerech naměřeno využití 15,5 % paměti. U 1021 spuštěných kontejnerů pak bylo zatížení navýšeno na hodnotu 75,6 %. Maximální využití paměti a následné ustálení grafické závislosti by nastalo přibližně u 1300 spuštěných kontejnerů.

Následně bylo provedeno měření doby, za kterou je vytvořen a spuštěn daný kontejner a měření prodlevy, která nastává při určitém zatížení hostitelského serveru. Realizace těchto měření je pak zobrazena jako Graf 14.

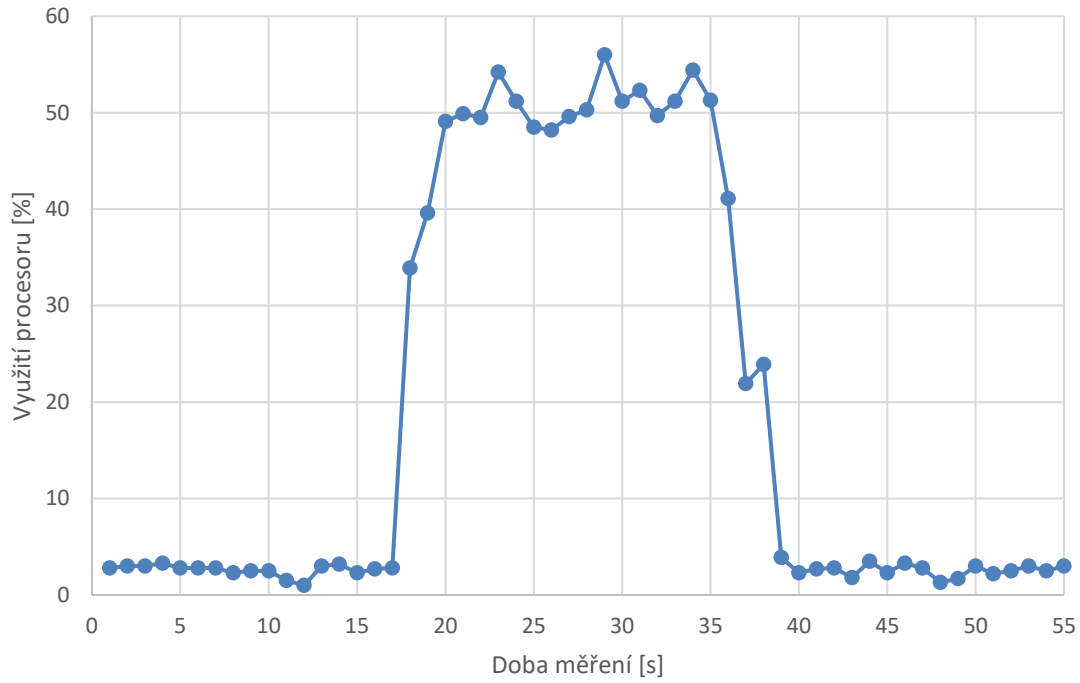


Graf 14 Odezvy při zvyšujícím se počtu kontejnerů s upraveným HW

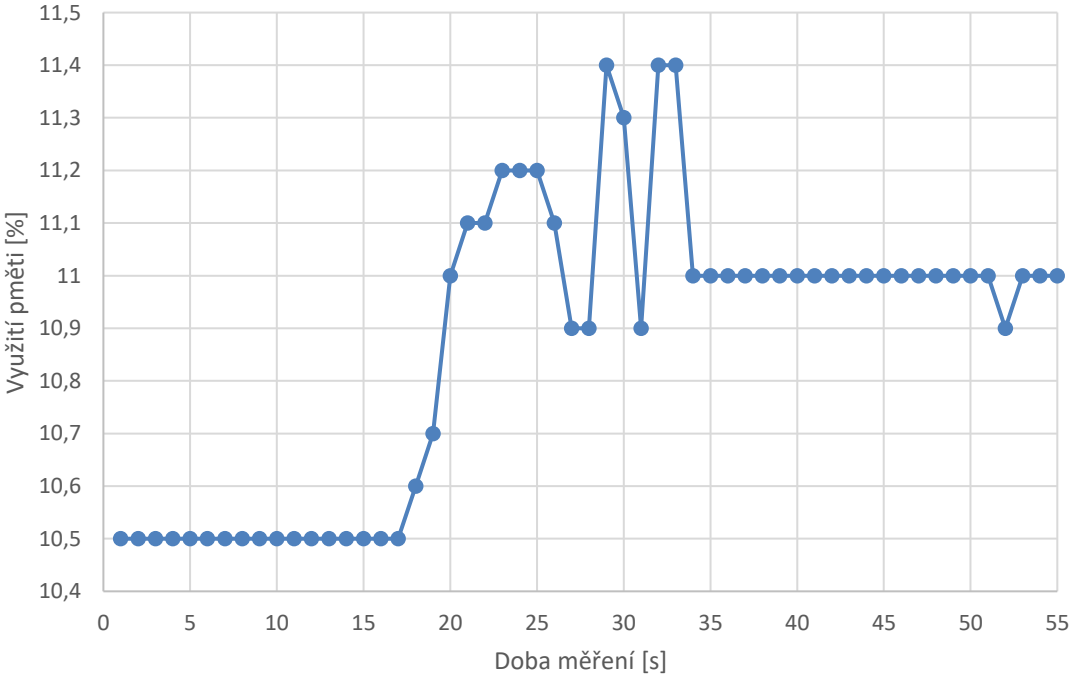
Grafická charakteristika *Navázání spojení* zobrazuje dobu, kterou trvá navázání spojení klient – server v rámci nástroje Ansible, v případě, že není spuštěn ani jeden kontejner je spojení navázáno přibližně za 0,75 s. Tato doba však roste lineárně se zvyšujícím se počtem vytvořených a spuštěných kontejnerů, kdy při 500 spuštěných kontejnerech se doba navázání spojení zvýšila na hodnotu 7,47 s, v případě, kdy je spuštěno 1021 kontejnerů je doba navázání spojení přibližně 15,7 sekund. Při porovnání hodnot, které byly naměřeny v rámci výchozích parametrů hardwaru, jsou doby navázání spojení téměř identické, s upravenými parametry hardwaru tak bylo možné realizovat větší počet měření. Doba navázání spojení tak není závislá na především na hardwaru, ale na počtu již běžících procesů v rámci hostitelského serveru. Druhá závislost *Vytvoření i-tého kontejneru* pak zobrazuje dobu kterou trvá vytvoření a spuštění daného kontejneru v rámci spuštěného playbooku. Vytvoření jednoho kontejneru trvá přibližně 0,66 s, 500. kontejner trvá vytvořit 1,43 s, 1021. kontejner je vytvořen a spuštěn přibližně za 2,15 s. Zatímco v případě upravených parametrů hardwaru trvalo vytvoření a spuštění kontejnerů v rámci playbooku 0,66-2,15 s, tak u výchozích parametrů hardwaru hostitelského serveru trvalo vytvoření a spuštění daných kontejnerů v rozmezí 0,8-6,2 s.

8.3.3 Měření zátěže

Pro měření bylo vytvořeno 50 klientských kontejnerů plus kontejner představující služby, tak aby bylo možné porovnat měření s výchozím a upraveným HW.

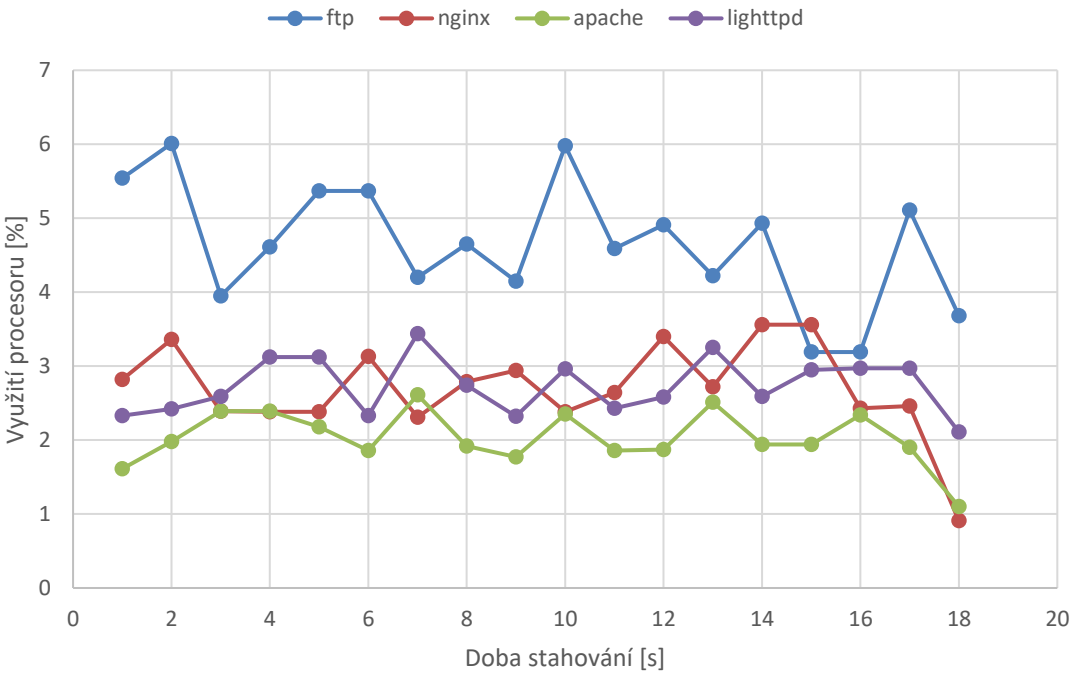


Graf 15 Zatížení procesoru při stahování s upraveným HW

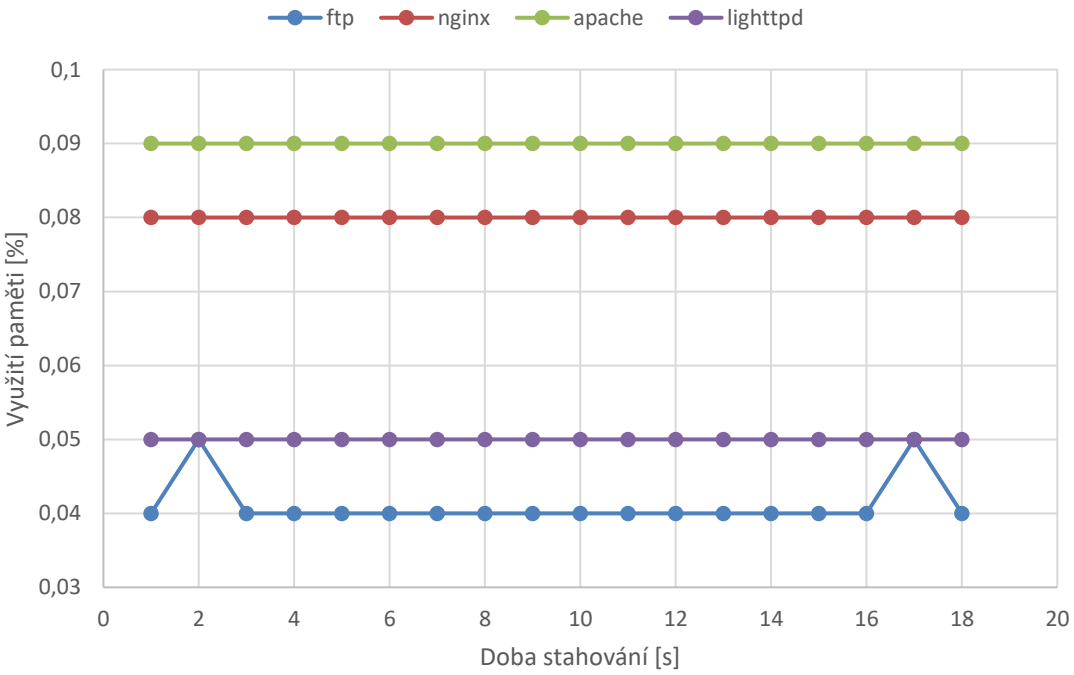


Graf 16 Zatížení paměti při stahování s upraveným HW

Využití procesoru se během stahování pohybuje v rozmezí 50-60 %, využití pamětí je pak při stahování v rozmezí 11-11,4 %



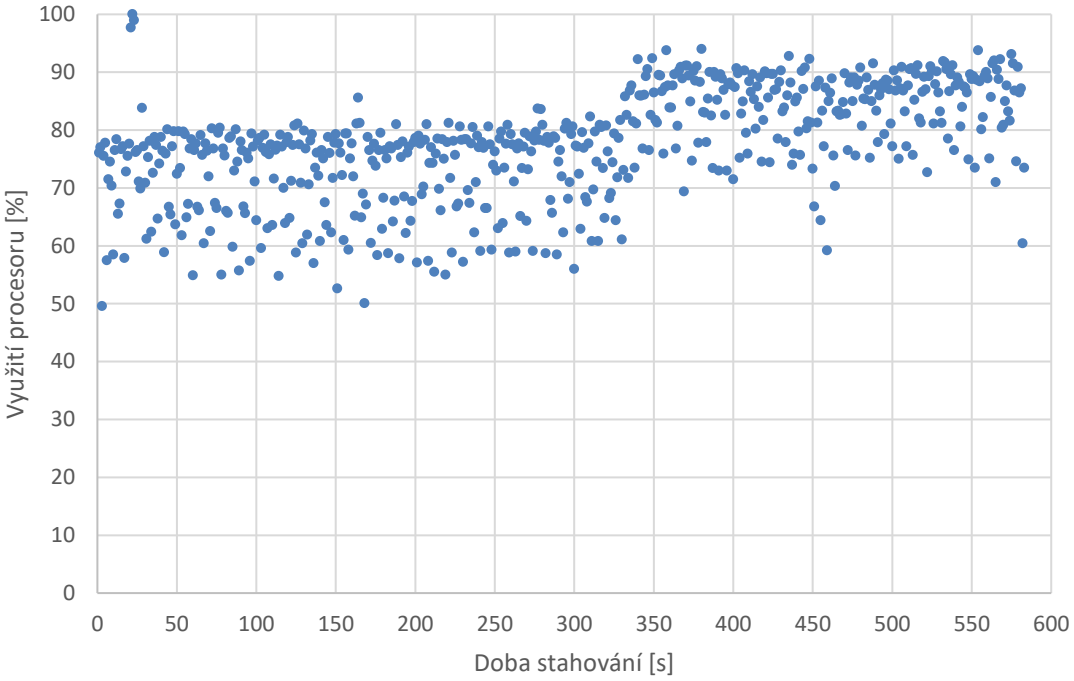
Graf 17 Využití procesoru jednotlivými službami s upraveným HW



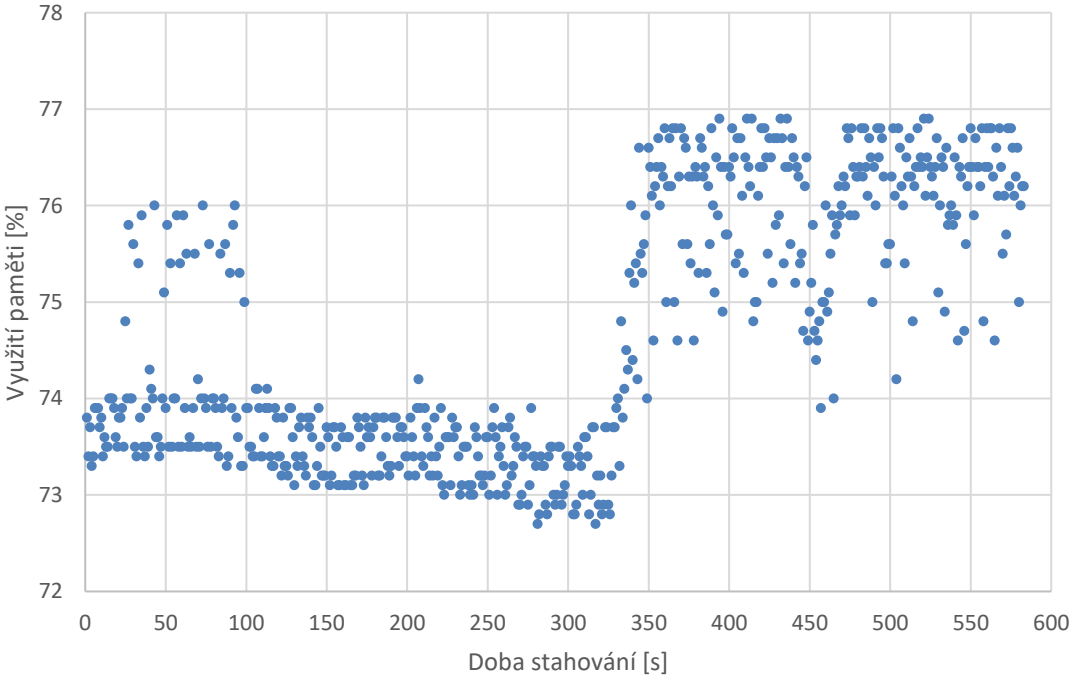
Graf 18 Využití paměti jednotlivými službami s upraveným HW

Oproti předchozímu měření s výchozími parametry došlo ke snížení doby stahování, a to ze 24-26 sekund na 18 sekund. V případě kontejnerů se službami dochází k ustálení využití paměti RAM.

Další měření zátěže bude realizováno prostřednictvím kontejneru s danou službou (FTP) a vytvořením 1 000 klientských kontejnerů.

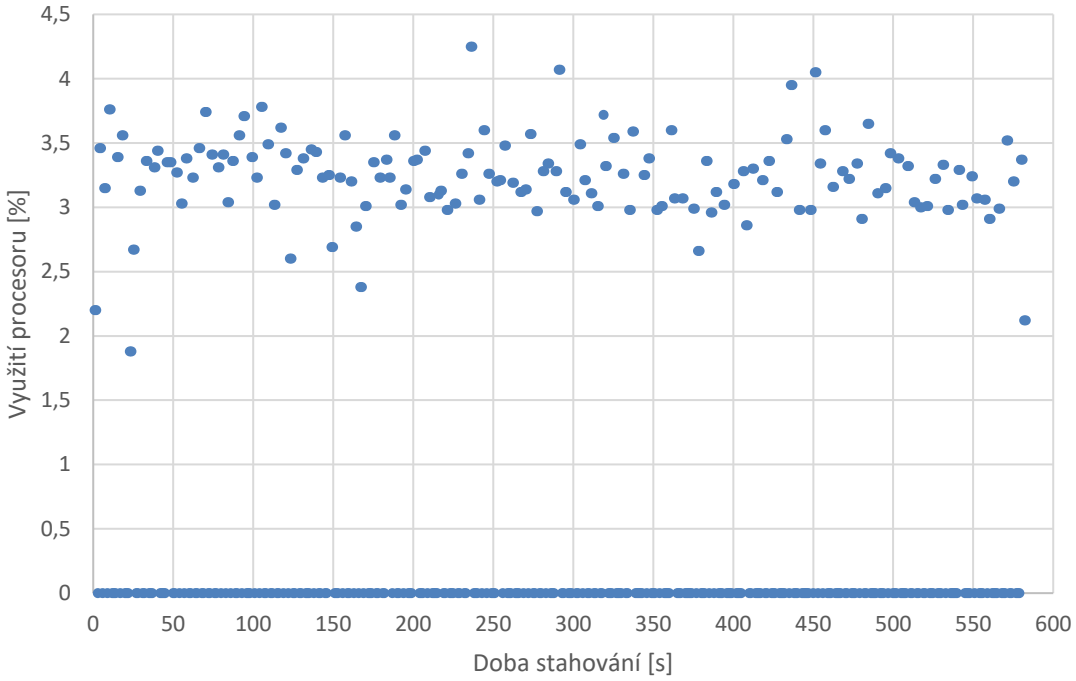


Graf 19 Využití procesoru při stahování 1 000 klientských kontejnerů ze serveru

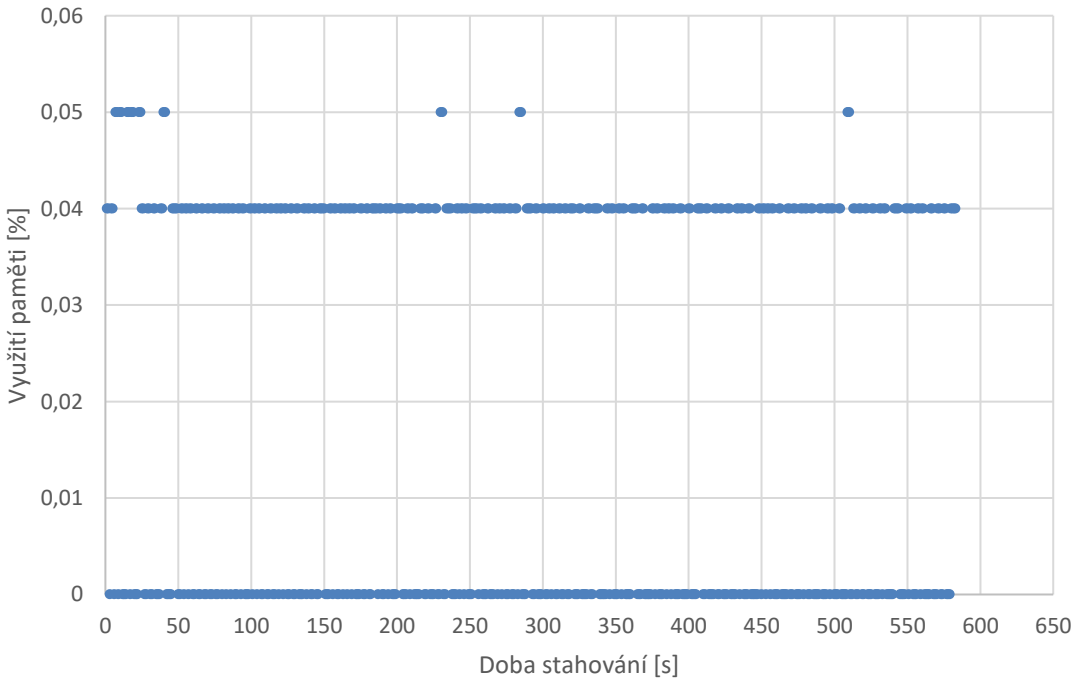


Graf 20 Využití paměti při stahování 1 000 klientských kontejnerů ze serveru

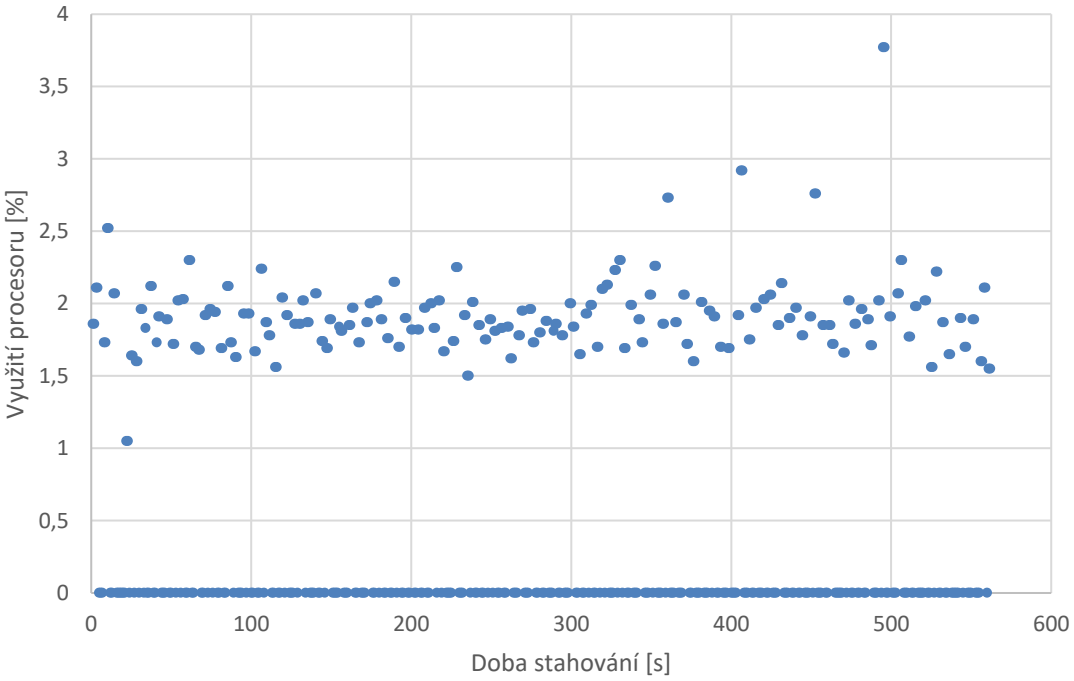
Měření s 1 000 vytvořených kontejnerů bylo realizováno pro všechny služby. Následující grafy zobrazují využití procesoru a pamětí jednotlivými kontejnery se službami.



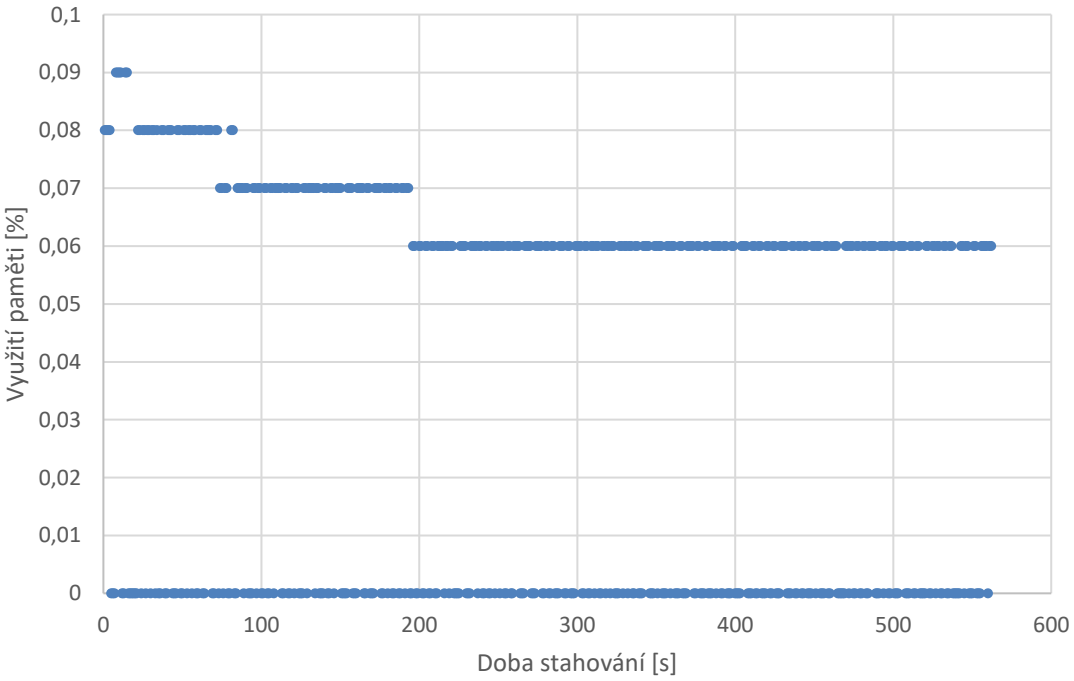
Graf 21 Využití procesoru FTP kontejnerem při stahování



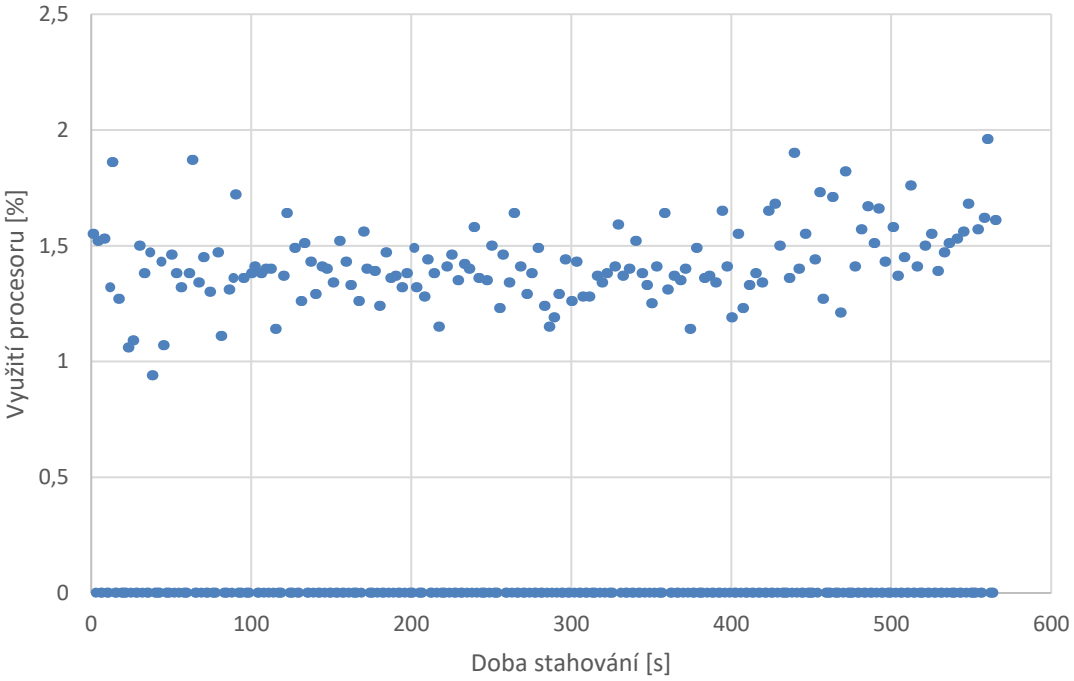
Graf 22 Využití paměti FTP kontejnerem při stahování



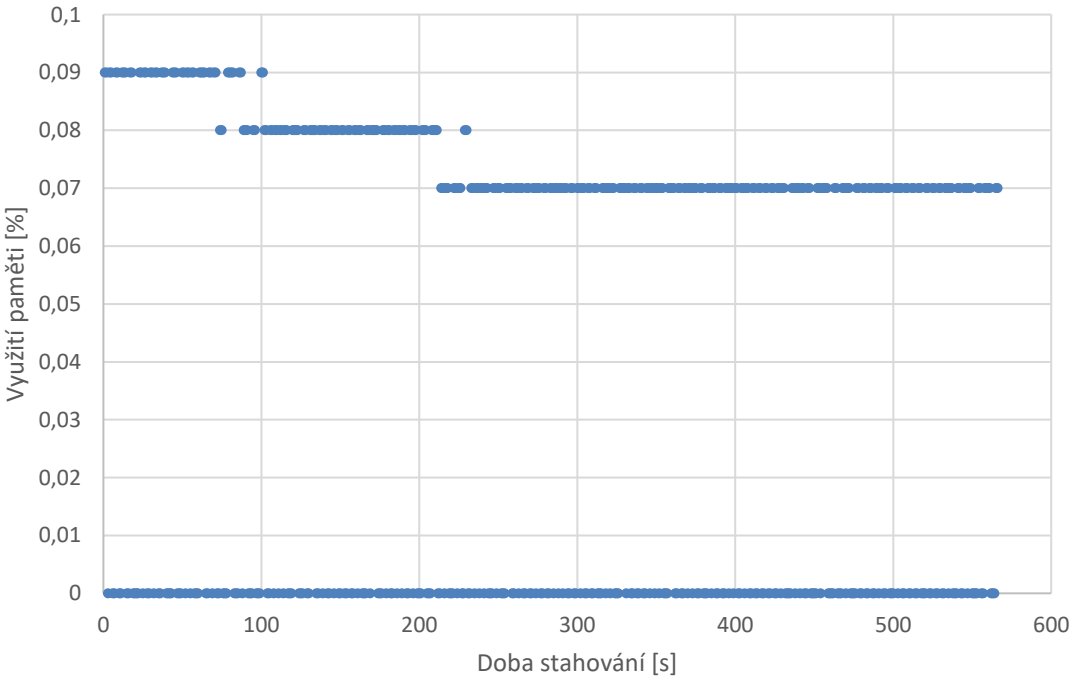
Graf 23 Využití procesoru Nginx kontejnerem při stahování



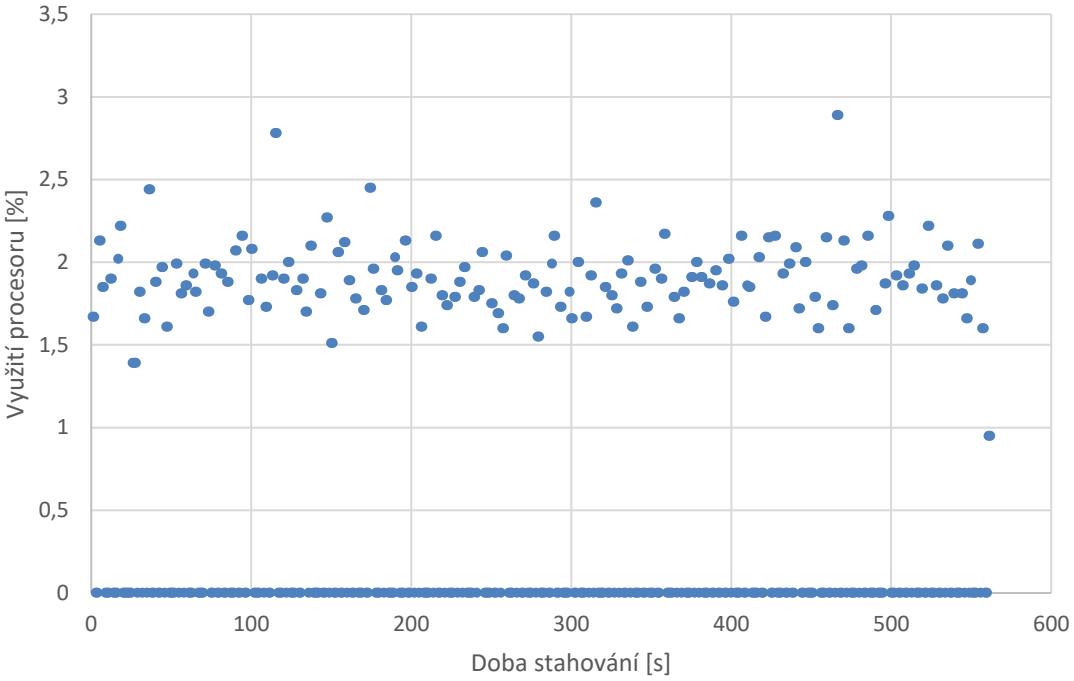
Graf 24 Využití paměti Nginx kontejnerem při stahování



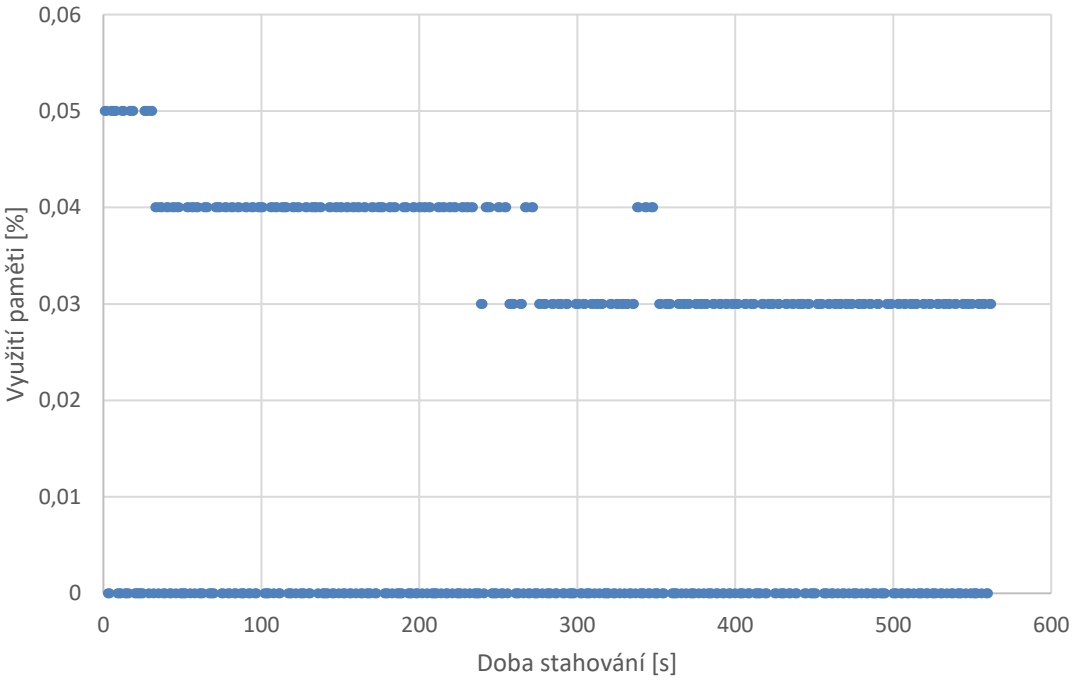
Graf 25 Využití procesoru Apache kontejnerem při stahování



Graf 26 Využití paměti Apache kontejnerem při stahování



Graf 27 Využití procesoru Lighttpd kontejnerem při stahování



Graf 28 Využití paměti Lighttpd kontejnerem při stahování

8.4 Vyhodnocení měření

V rámci kapitoly 8 bylo realizováno několik druhů měření, a to měření naprázdno, měření se spuštěnými klienty, měření odezvy při navázání spojení a vytvoření i-tého kontejneru, měření zátěže s určitým počtem spuštěných kontejnerů, a nakonec využití procesoru a paměti jednotlivými službami.

V případě měření naprázdno bylo změřeno zatížení HW při spuštění měřících aplikací jak u výchozích, tak u upravených parametrů HW. Z naměřených hodnot bylo zjištěno, že po spuštění těchto aplikací dochází v případě procesoru ke krátkému skokovému nárostu jeho využití a následnému ustálení, z tohoto důvodu bylo při měření a následném zpracování výsledů zanedbáno prvních 5 hodnot. V rámci paměti je pak v případě grafické závislosti s výchozími a upravenými parametry HW možné vidět postupný nárust využití při postupném spouštění jednotlivých měřících aplikací. V případě výchozích parametrů se jedná o nárust 15 %, v případě upravených parametrů pak o 1,5 %.

U měření zatížení HW při spuštění určitého počtu kontejnerů s výchozími a upravenými parametry bylo zjištěno, že procesor je ovlivňován zejména měřícími aplikacemi, které se zvyšujícím se počtem měřených a zpracovávaných dat zatěžují procesor téměř lineárně. V případě zatížení paměti se jedná o lineární průběhy charakteristik, v případě výchozích parametrů pak bylo možné vidět postupné ustálení tohoto lineárního průběhu, které bylo způsobeno nedostatkem paměti, kdy v tomto případě nástroj Docker snižuje počet aktivních kontejnerů v daný okamžik.

V případě měření odezvy při navázání spojení a vytvoření i-tého kontejneru v rámci daného playbooku byly získány charakteristiky s lineárními průběhy, kdy jednotlivé odezvy se zvyšují se zvyšujícím se počtem vytvořených a spuštěných kontejnerů. Doba odezvy je v jednotlivých případech ovlivněna i hodnotami přiděleného HW.

Na závěr bylo provedeno měření zátěže pro 50 spuštěných kontejnerů, jak pro výchozí, tak upravené hodnoty HW. V případě spuštění stahování došlo jak u procesoru, tak u pamětí k jejich okamžitému zatížení a následnému snížení po dokončení stahování. V případě výchozích hodnot HW trvalo stahování přibližně 25 s. Po upravení parametrů HW se doba stahování zkrátila přibližně o 7 s, došlo taky k ustálení využití pamětí v případě jednotlivých kontejnerů se službami, v případě těchto kontejnerů pak docházelo k většímu využití procesoru oproti výchozím parametrům, HW. S těmito upravenými parametry HW bylo možné realizovat měření zátěže i pro 1000 spuštěných kontejnerů. V rámci grafů zobrazující využití procesoru a paměti je možné vidět v daný okamžik i nulové hodnoty. Tyto hodnoty jsou způsobeny tím, že při takovémto množství spuštěných kontejnerů docházelo k tomu, že v daný okamžik nebyly některé kontejnery aktivní včetně kontejnerů představující služby.

ZÁVĚR

Cílem této práce bylo otestovat dostupné nástroje, umožňují automatizovat virtuální infrastrukturu, tyto nástroje porovnat a vybrat nejvhodnější nástroj, který bude dále využíván v rámci závěrečné práce.

Nejdříve byly jednotlivé nástroje nainstalovány a nakonfigurovány tak, aby bylo možné komunikovat a následně konfigurovat vzdálený server v rámci lokální sítě. V rámci nástrojů Ansible a Terraform bylo nutné realizovat SSH spojení, v případě nástroje Puppet se jednalo o vytvoření a potvrzení certifikátu.

Na základě těchto výsledků však bylo určeno, že nejvhodnější by bylo použití nástrojů Ansible či Terraform zejména z důvodu toho, že oproti nástroji Puppet zde odpadá nutnost instalace nástroje i na konfigurovaný server. Aby bylo možné tedy vybrat jeden z nástrojů, byl následně vytvořen konfigurační soubor pro instalaci a konfiguraci publikačního systému WordPress s využitím Nginx serveru, na kterém WordPress poběží. Tato instalace potvrdila využití jednotlivých nástrojů, zatímco Ansible je spíše určen k instalacím a konfiguracím serveru, Terraform je pak vhodnější k vyváření serverů či služeb neboli popsat, jak by měla infrastruktura vypadat a následně ji tak vytvořit. Ansible však nabízí nejrůznější množství modulů ať již systémových tak i modulů pro různé služby. Z tohoto důvodu je nástroj Ansible nejvhodnější a nejuniverzálnější nástroj pro automatizaci virtuální infrastruktury.

Pomocí tohoto nástroje byly následně vytvořeny jednotlivé scénáře umožňující instalaci a konfiguraci potřebných nástrojů v tomto případě se jednalo o instalaci nástroje Docker a vytvoření potřebných sítí. Dále zde byly implementovány scénáře pro správu hostitelského serveru a nástroje Docker v podobě výpisu dostupných kontejnerů, imagů či sítí a následné možnosti vytvoření a spuštění kontejnerů realizující klienty či služby. Nakonec byly vytvořeny potřebné scénáře a skripty pro realizaci měření. Všechny tyto vytvořené scénáře byly sjednoceny prostřednictvím grafického rozhraní a nebylo tak nutné pro jejich spuštění zadávat příslušné příkazy.

Nakonec bylo realizováno měření, které se skládalo ze tří následujících částí, a to z měření naprázdno, kde bylo měřeno zejména jaký vliv na měření budou mít samotné aplikace realizující měření. Další částí je měření s výchozími parametry HW, třetí částí je pak měření s upravenými parametry HW, kde bylo provedeno měření zátěže a bylo možné tak porovnat zatížení procesoru a paměti RAM při rozdílných parametrech přiděleného hardwaru.

LITERATURA

[1] POMAZAL, Jiří. Virtualizace v kostce. *SystemOnLine* [online]. Brno: CCB, spol. s r. o., 2010 [cit. 2017-12-08]. Dostupné z: https://www.systemonline.cz/clanky/virtualizace-v-kostce.htm

[2] TIŠNOVSKÝ, Pavel. Sálové počítače firmy IBM: IBM 704 – sálový počítač... *Root.cz* [online]. Praha: Internet Info, s.r.o., 2009 [cit. 2017-12-08]. Dostupné z: https://www.root.cz/clanky/salove-pocitace-firmy-ibm/#k05

[3] About Us. *Vmware* [online]. Palo Alto: VMware, Inc, 2019 [cit. 2019-12-16]. Dostupné z: https://www.vmware.com/company.html

[4] News (older entries). *VirtualBox* [online]. Redwood Shores: Oracle Corporation, 2019 [cit. 2019-12-16]. Dostupné z: https://www.virtualbox.org/wiki/News_pre_4_3

[5] KILIÁN, Karel. Virtualizace aplikací je s Evalaze... *Cnews.cz* [online]. Praha: Mladá fronta a.s., 2013 [cit. 2017-12-08]. Dostupné z: https://www.cnews.cz/virtualizace-aplikaci-je-s-evalaze-brnkacka-chrante-sva-windows/

[6] RUEST, Danielle a Nelson RUEST. *Virtualizace: podrobný průvodce*. Vyd. 1. Brno: Computer Press, 2010. ISBN 978-80-251-2676-9.

[7] SEDLÁK, Jan. Aplikace v kontejneru. Co je Docker... *E15.cz* [online]. Praha: CN Invest a.s, 2015 [cit. 2017-12-08]. Dostupné z: http://e-svet.e15.cz/it-byznys/aplikace-v-kontejneru-co-je-docker-a-proc-ho-vsichni-chteji-1157856

[8] Docker libcontainer unifies Linux container powers. *ZDNet* [online]. San Francisco: CBS Interactive, 2019 [cit. 2019-12-16]. Dostupné z: https://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/

[9] Co to vlastně je RAID a jaké je jeho užití?. *DATAHELP* [online]. Praha: Aira GROUP, s.r.o., 2019 [cit. 2019-12-16]. Dostupné z: https://www.datahelp.cz/clanky/co-to-vlastne-je-raid-a-jake-je-jeho-uziti

[10] ROUSE, Margaret. RAID (redundant array of independent disks). *TechTarget* [online]. Newton: TechTarget, c2000-2019 [cit. 2019-12-16]. Dostupné z: https://searchstorage.techtarget.com/definition/RAID

[11] BOUŠKA, Petr. VLAN - Virtual Local Area Network: Co je to VLAN. *SAMURAJ-cz* [online]. \n: Samuraj, 2007 [cit. 2017-12-08]. Dostupné z: https://www.samuraj-cz.com/clanek/vlan-virtual-local-area-network/

[12] Infrastruktura. *IT SLOVNÍK.cz* [online]. Wien: IT-Slovnik.cz team, /n [cit. 2019-12-07]. Dostupné z: https://it-slovnik.cz/pojem/infrastruktura/?utm_source=cp&utm_medium=link&utm_campaign=cp

[13] ZAPLETAL, Lukáš. Klíčem k úspěšné správě IT infrastruktury je automatizace. *SystemOnLine* [online]. Brno: CCB, spol. s r. o., 2015 [cit. 2019-12-07]. Dostupné z: https://www.systemonline.cz/sprava-it/foreman-automatizuje-spravu-it.htm

[14] PILAŘ, Jan. Správa virtuální IT infrastruktury. *SystemOnLine* [online]. Brno: CCB, spol. s r. o., 2009 [cit. 2019-12-07]. Dostupné z: https://www.systemonline.cz/virtualizace/virtualizace-v-praxi-5.-dil.htm

[15] 25 things you should know about Red Hat. *Red Hat* [online]. Raleigh: Red Hat, Inc, 2019 [cit. 2019-12-16]. Dostupné z: https://www.redhat.com/en/blog/25-things-you-should-know-about-red-hat-0

[16] HORNIK, Patrik. Red Hat - open sourceová komunita nie je slepá. História a súčasnosť Red Hat-u. *DSL.sk* [online]. /n: DSL.sk, c2004-2005 [cit. 2019-12-16]. Dostupné z: http://www.dsl.sk/article.php?article=67&title=Red-Hat-komunita-nie-je-slepa

[17] IBM dokončila akvizici společnosti Red Hat. *HOSPODÁŘSKÉ NOVINY* [online]. Praha 8: Economia, a.s., 2019 [cit. 2019-12-07]. Dostupné z: https://ictrevue.ihned.cz/c1-66607210-ibm-dokoncila-akvizici-spolecnosti-red-hat

[18] The Origins of Ansible. *Red Hat Ansible* [online]. Raleigh: Red Hat, Inc., 2019 [cit. 2019-12-16]. Dostupné z: https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible

[19] Red Hat to Acquire IT Automation and DevOps Leader Ansible. *Red Hat Ansible* [online]. Raleigh: Red Hat, Inc., 2019 [cit. 2019-12-16]. Dostupné z: https://www.redhat.com/en/about/press-releases/red-hat-acquire-it-automation-and-devops-leader-ansible#

[20] An Ansible Tutorial. *Servers for Hackers* [online]. Fideloper LLC, 2014 [cit. 2019-12-07]. Dostupné z: https://serversforhackers.com/c/an-ansible-tutorial

[21] Installation Guide. *Documentation* [online]. /n: Red Hat, Inc., 2019 [cit. 2019-12-07]. Dostupné z: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

[22] About Us. *HashiCorp* [online]. San Francisco: HashiCorp, Inc., 2018 [cit. 2019-12-16]. Dostupné z: https://www.hashicorp.com/about/

[23] Cloud Infrastructure Automation. *HashiCorp* [online]. San Francisco: HashiCorp, Inc., 2018 [cit. 2019-12-16]. Dostupné z: https://www.hashicorp.com/products/terraform/

[24] Installing Terraform. *Terraform* [online]. San Francisco: HashiCorp, Inc., 2018 [cit. 2019-12-07]. Dostupné z: https://learn.hashicorp.com/terraform/getting-started/install.html

[25] Puppet Enterprise. *Puppet* [online]. Portland: Puppet, Inc., 2019 [cit. 2019-12-17]. Dostupné z: https://puppet.com/products/puppet-enterprise/

[26] Open source projects at Puppet. *Puppet* [online]. Portland: Puppet, Inc., 2019 [cit. 2019-12-17]. Dostupné z: https://puppet.com/open-source/

[27] The Puppet language. *Puppet* [online]. Portland: Puppet, Inc., 2019 [cit. 2019-12-17]. Dostupné z: https://puppet.com/docs/puppet/latest/puppet_language.html

[28] Puppet. *Ubuntu documentation* [online]. /n: Ubuntu Documentation Team, /n [cit. 2019-12-17]. Dostupné z: https://help.ubuntu.com/lts/serverguide/puppet.html

[29] Puppet.conf: The main config file. *Puppet* [online]. Portland: Puppet, Inc., 2019 [cit. 2019-12-17]. Dostupné z: https://puppet.com/docs/puppet/latest/config_file_main.html

[30] Puppet Tutorial. *IntelliPaat* [online]. India: intellipaat.com, 2019 [cit. 2019-12-07]. Dostupné z: https://intellipaat.com/blog/tutorial/devops-tutorial/puppet-tutorial/

[31] Nginx. *NGINX* [online]. San Francisco: Nginx, Inc., 2018 [cit. 2019-12-17]. Dostupné z: https://nginx.org/en/

[32] KRČMÁŘ, Petr. Nginx v roli web serveru. *PETR KRČMÁŘ* [online]. Česko: Creative Commons, 2015 [cit. 2019-12-17]. Dostupné z: https://www.petrkrcmar.cz/prednasky/Nginx_2013.pdf

[33] HOCHSTEIN, Lorin. *Ansible: up and running*. Online. Beijing: O'Reilly, 2015. ISBN 978-1-491-91532-5.

[34] File Provisioner. *Terraform* [online]. San Francisco: HashiCorp, Inc., 2018 [cit. 2019-12-17]. Dostupné z: https://www.terraform.io/docs/provisioners/file.html

[35] Remote-exec Provisioner. *Terraform* [online]. San Francisco: HashiCorp, Inc., 2018 [cit. 2019-12-17]. Dostupné z: https://www.terraform.io/docs/provisioners/remote-exec.html

[36] ARUNDEL, John. *Puppet 5 Beginner's Guide: Go from Newbie to Pro with Puppet 5*. 3rd ed. Birmingham: Published by Packt Publishing Ltd., 2017. ISBN 978-1-78847-290-6.

[37] How To Install WordPress with Nginx on Ubuntu 14.04. *Digital Ocean* [online]. New York: DigitalOcean, LLC., 2019 [cit. 2019-12-17]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-with-nginx-on-ubuntu-14-04

[38] How to install MySQL on Ubuntu 18.04/19.04. *TubeMint* [online]. /n: TubeMint, 2019 [cit. 2019-12-17]. Dostupné z: https://tubemint.com/how-to-install-mysql-on-ubuntu/

[39] 4.2.2.1 Using Options on the Command Line. *MySQL* [online]. Redwood Shores: Oracle Corporation, 2019 [cit. 2019-12-17]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/command-line-options.html

[40] ROUSE, Margaret. FTP (File Transfer Protocol). *TechTarget* [online]. Newton: TechTarget, c2000-2020 [cit. 2020-04-24]. Dostupné z: https://searchnetworking.techtarget.com/definition/File-Transfer-Protocol-FTP

[41] What is the Difference Between Active and Passive FTP?. *Titan* [online]. Annapolis: South River Technologies, Inc., 2020 [cit. 2020-04-24]. Dostupné z: https://titanftp.com/2018/08/23/what-is-the-difference-between-active-and-passive-ftp/

[42] Vsftpd. *Ubuntu documentation* [online]. /n: Ubuntu Documentation Team, /n [cit. 2020-04-24]. Dostupné z: https://help.ubuntu.com/community/vsftpd

[43] ZAPLETAL, Lukáš. Protokol HTTP 1.1 pod lupou. *ROOT.CZ* [online]. Praha: Internet Info, c1998-2020 [cit. 2020-04-24]. Dostupné z: https://www.root.cz/clanky/protokol-http-1-1-pod-lupou/

[44] FERSCHMANN, Petr. Bezpečnost na webu – přehled útoků na webové aplikace. *Zdroják.cz* [online]. Praha: Devel.cz Lab, b. r. [cit. 2020-04-24]. Dostupné z: https://www.zdrojak.cz/clanky/prehled-utoku-na-webove-aplikace/

[45] JANÍK, David. Apache vs Nginx. *Váš hosting* [online]. Nymburk: Váš Hosting, 2019 [cit. 2020-04-24]. Dostupné z: https://www.vas-hosting.cz/blog-apache-vs-nginx

[46] N., Aldwin. What is NGINX? How does it work?. *Hostinger* [online]. Kaunas: hostinger.com, c2004-2020 [cit. 2020-04-24]. Dostupné z: https://www.hostinger.com/tutorials/what-is-nginx

[47] PARAJULI, Sagar. The Battle of the Web Servers:Apache Vs Nginx Vs Lighttpd. *Detechter* [online]. New York: DigitalOcean, LLC, 2020 [cit. 2020-04-24]. Dostupné z: https://detechter.com/the-battle-of-the-web-servers-apache-vs-nginx-vs-lighttpd-2/

[48] ŠTRAUCH, Adam. Lighttpd: lehký webserver. *ROOT.CZ* [online]. Praha: Internet Info, c1998-2020 [cit. 2020-04-24]. Dostupné z: https://www.root.cz/clanky/lighttpd-lehky-webserver/

[49] GUPTA, Kitty. Lighttpd vs Nginx: What You Need to Know Read more at: https://www.freelancinggig.com/blog/2017/05/25/lighttpd-vs-nginx-need-know/. *FreelancingigGig* [online]. Tampa: FreelancingGig, 2020 [cit. 2020-04-24]. Dostupné z: https://www.freelancinggig.com/blog/2017/05/25/lighttpd-vs-nginx-need-know/

Seznam symbolů, veličin a zkratek

AMD-V	-	AMD virtualization
C10K	-	10 housand clients problem
CLI	-	Command-line interface
CPU	-	Central processing unit
FTP	-	File Transfer Protocol
HCL	-	HashiCorp Configuration Language
HDD	-	Hard Disk Drive
HTML	-	Hypertext Markup Language
HTTP	-	Hypertext Transfer Protocol
HTTPS	-	Hypertext Transfer Protocol Secure
HW	-	Hardware
IMAP	-	Internet Message Acces Protokol
IP	-	Internet Protocol
MAC	-	Media Access Control
MPM	-	MultiProcessing
PHP	-	Hypertext Preprocessor
POP3	-	Post Office Protocol
RAID	-	Redundant Array of Independent Disks
RAM	-	Random-acces memory
SMTP	-	Simple Mail Transfer Protocol
SQL	-	Structured Query Language
SSH	-	Secure Shell
SSL	-	Secure Sockets Layer
URL	-	Uniform Resource Locator
USB	-	Universal Serial Bus
UTF-8	-	UCS/Unicode Transformation Format
VT-x	-	Intel virtualization
YAML	-	YAML Ain't Markup Language

Seznam příloh