Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Instalace a zabezpečení serverových služeb za využití kontejnerů LXC a Docker

Bc. Michal Kružík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Michal Kružík

Informatika

Název práce

Instalace a zabezpečení serverových služeb za využití kontejnerů LXC a Docker

Název anglicky

Installation and secure of server services using LXC and Docker containers

---

Cíle práce

Hlavním cílem je vytvoření plně funkční sady serverových služeb zahrnujících databázový server, webserver a mailserver za využití kontejnerových technologií Docker a LXC. Dílčím cílem je porovnání náročnosti nastavených služeb z hlediska systémových nároků, bezpečnostni a časové náročnosti oproti sadě služeb bez využití kontejnerů.

Metodika

Teoretická část bude obsahovat popis a charakteristiku kontejnerových technologií, jejich výhody a nevýhody oproti plně virtualizovanému a nevirtualizovanému prostředí. Bude založena na studiu odborné a vědecké literatury. Dále budou charakterizovány kontejnery LXC a Docker.

Praktická část bude obsahovat instalace a bezpečnou konfiguraci vybraných serverových služeb. Testování bude probíhat ve dvou stejných virtualizovaných prostředích. Jedno prostředí bude využito pro služby běžící bez kontejnerů přímo na hostovaném operačním systému a druhé bude využívat kontejnerové technologie.

V závěrečné části budou vybrané služby porovnány v obou prostředích s ohledem na paměťovou, diskovou a výpočetní náročnost a bude zhodnocena vhodnost jejich nasazení v nefiremním prostředí.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

linux, server, lxd, docker, kontejnerové technologie

Doporučené zdroje informací

ALIBI, Mohamed a Roy, Bhaskarjyoti. Mastering CentOS 7 Linux Server. Packt Publishing, 2016. ISBN: 978-1785282393

BOWEN, Rich. Apache Cookbook: Solutions and Examples for Apache Administrators. O'Reilly Media, 2008. ISBN: 978-0596529949

DENT, D. Kyle, Postfix: The Definitive Guide: A Secure and Easy-to-Use MTA for UNIX. O'Reilly Media, 2003. ISBN: 978-0596002121

DUBOIS, Paul. MySQL Cookbook: Solutions for Database Developers and Administrators. O'Reilly Media, 2014. ISBN: 978-1449374020

GOASGUEN, Sébastien. Docker Cookbook: Solutions and Examples for Building Distributed Applications. O'Reilly Media, 2015. ISBN: 978-1491919712

NICKOLOFF, Jeff. Docker in Action. Manning Publications, 2016. ISBN: 978-1633430235

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Alexandr Vasilenko, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 30. 10. 2017

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2017

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 27. 03. 2018

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Instalace a zabezpečení serverových služeb za využití kontejnerů LXC a Docker" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 1. března 2018 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce Ing. Alexandru Vasilenkovi, Ph.D. za odborné konzultace a ochotu vést tuto práci. Dále děkuji svým rodičům a přátelům, kteří mě vytrvale podporují po dobu mých studií.

Instalace a zabezpečení serverových služeb za využití kontejnerů LXC a Docker

---

Installation and secure of server services using LXC and Docker containers

Souhrn

Tato diplomová práce je reakcí na snahu o vývoj a provoz serverových služeb v izolovaném prostředí z důvodu bezpečnosti, testování a přenositelnosti mezi servery. Popisuje kontejnerové technologie LXC a Docker, možnosti jejich využití, výhody, nevýhody a omezení.

Cílem této práce je instalace a konfigurace serverových linuxových služeb ve variantách bez využití kontejnerů, za využití kontejnerů LXC a Docker. Dílčím cílem je jejich porovnání z hlediska výkonu a náročnosti na systémové prostředky.

Summary

This project is based on the development and maintenance of server-side services in isolated environment for security, testing and migrating purposes. It describes LXC and Docker container technologies, their use cases, advantages and disadvantages along with their limitations.

The primary focus of this project will be the installation and configuration of server linux services, with and without containers, as well as solution comparison.

Klíčová slova:

linux, server, lxc, lxd, docker, kontejnerové technologie, bezpečnost, mailserver, webserver

Keywords:

linux, server, lxc, lxd, docker, container technologies, security, mailserver, webserver

Obsah

1 Úvod

Ačkoliv je serverová virtualizace již několik let využívána firmami k optimalizaci zátěže serverů, jejich rozdělení pro klienty a k jejich izolaci, má své omezení. Pro izolaci jednotlivých aplikací, či jejich souvisejících skupin se nehodí. Vyžaduje pokročilé znalosti virtualizačních nástrojů, výkonnější hardware a obvykle nemalé množství místa na samotné virtualizované servery v řádech jednotek až desítek gigabajtů.

Z těchto důvodů se začaly objevovat snahy o virtualizaci software na úrovni operačního systému. Ze začátku neposkytovaly virtualizační nástroje dostatečnou bezpečnost a měly znatelný vliv na výkon. Situace v současné době je již znatelně lepší a izolace procesů poskytuje dostatečnou úroveň bezpečnosti a efektivity. [1] Díky tomu mohly vzniknout nástroje pro vytváření kontejnerů, které sdílí s operačním systémem jen jádro a je tak možné kontejnerovaný software jednoduše migrovat mezi servery bez nutnosti jej znovu nastavovat. Mezi tyto nástroje patří LXC, respektive LXD, a Docker.

Tato práce je věnována instalaci a zabezpečenému nastavení linuxového serveru se službami pro webserver a mailserver za využití kontejnerových technologií LXC a Docker. Cílem je dosáhnout efektivní izolace služeb od hostujícího operačního systému a sebe navzájem. Nabízí řešení aplikace kontejnerů pro serverové služby, s cílem zlepšit jejich přenositelnost a bezpečnost s minimálním vlivem na výkon a náročnost správy.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem je vytvoření plně funkční sady serverových služeb zahrnujících databázový server, webserver a mailserver za využití kontejnerových technologií Docker a LXC. Dílčím cílem je porovnání náročnosti nastavených služeb z hlediska systémových nároků, bezpečnosti a časové náročnosti konfigurace oproti sadě služeb bez využití kontejnerů.

2.2 Metodika

Metodika řešené práce bude založena na analyticko-syntetickém přístupu, studiu odborné a vědecké literatury a zkušenostech autora s použitými technologiemi. Teoretická část bude obsahovat popis a charakteristiku kontejnerových technologií, jejich výhody a nevýhody oproti plně virtualizovanému a nevirtualizovanému prostředí. Dále budou charakterizovány kontejnery LXC a Docker.

Praktická část bude obsahovat postup instalace a bezpečnou konfiguraci vybraných serverových služeb. Testování bude probíhat ve třech stejných virtualizovaných prostředích. Jedno prostředí bude využito pro služby běžící bez kontejnerů přímo na hostovaném operačním systému a dvě budou využívat kontejnerové technologie.

V závěrečné části budou vybrané služby porovnány v obou prostředích s ohledem na paměťovou, diskovou a výpočetní náročnost a bude zhodnocena vhodnost jejich nasazení v nekomerčním prostředí.

3 Přehled řešené problematiky

Kontejnerové a virtualizační technologie mají v současné době široké uplatnění. Umožňují bezpečnější a nezávislejší běh procesů a služeb a zlepšují jejich přenositelnost. Těchto výhod se využívá pro poskytování služeb virtuálních serverů, automatizovaného testování software, zabezpečení a izolování serverových služeb atp.

Je však důležité odlišovat virtualizaci celého operačního systému a virtualizaci na úrovni OS, která využívá kontejnerové technologie jako je LXC a Docker. Ty mají odlišné požadavky, omezení a jsou vhodné na řešení různých úkolů.

3.1 Kontejnerové technologie

Kontejnerovými technologiemi, respektive kontejnerovou virtualizací, je míněno vytvoření vzájemně oddělených prostředí (kontejnerů) v rámci jednoho operačního systému. Toto oddělení může zahrnovat izolaci souborového systému, paměti, sítě, popřípadě dalších dostupných prostředků. [2]

3.1.1 Historie

První koncept kontejnerů a izolace aplikací byl implementován v systému UNIX v roce 1979 spolu s příkazem *chroot* – change root. Tento příkaz změnil kořenovou složku procesu a omezil mu tak přístup k souborovému systému. Nejednalo se však o bezpečnostní řešení. Izolovaný proces mohl s dostatečnými právy opakovaně změnit kořenovou složku a připojovat si vzdálené složky. Chrooted proces též mohl bez omezení nadále využívat veškeré dostupné počítačové zdroje a přímo interagovat s ostatními běžícími procesy.

V roce 2000 byl do linuxové distribuce FreeBSD přidán příkaz *jail*. Jednalo se o začátek snahy o skutečnou izolaci procesů a služeb s ohledem na bezpečnost. Vznikl díky snaze hostingové firmy R&D Associates o vytvoření plně izolovaných prostředí pro své služby a klienty. BSD jail umožňuje omezit procesu přístup k souborům a službám hostujícího operačního systému. „Uvězněný“ proces též nemá přístup k procesům vně

svého „vězení“, čímž došlo k posílení zabezpečení serveru. Takto omezené procesy sdílí veškeré počítačové zdroje jako je paměť a CPU, bez omezení.

Mezi lety 2001 a 2005 vzniklo několik kontejnerových technologií (Linux VServer, Oracle Solaris Containers, Open VZ). Tyto technologie přinesly možnost omezení počítačových zdrojů, jednodušší zálohování a větší zabezpečení hostujícího počítače. Jejich nevýhodou byla nutnost úpravy jádra Linuxu, což omezovalo možnosti nasazení a přenositelnost.

Podpora kontejnerů na úrovní jádra přišla s řešením Process Containers od společnosti Google v roce 2006. Stejně jako předchozí řešení umožnovalo omezení zdrojů jako je CPU, paměť, přístup k disku a síti. Do samotného jádra Linuxu se dostalo řešení pod názvem Control Groups (cgroups) v roce 2007 ve verzi 2.6.24.

Control Groups byly využity spolu s linuxovými jmennými prostory (Linux namespaces) při tvorbě LXC, které bylo vydáno v roce 2008. Jednalo se o první skutečně plnohodnotné kontejnerové řešení, které bylo možné využít na neupraveném jádře Linuxu.

Google v roce 2013 uvolnil open-source verzi svého kontejnerového řešení pod zkratkou LMCTFY – Let Me Contain That For You. Cílem bylo vytvořit řešení s vysokou účinností, spolehlivostí, které je schopné optimálně spravovat velké množství sdílených zdrojů. LMCTFY bylo využito při tvorbě nástroje cAdvisor, které využívá systém Kubernetes, sloužící k automatizaci nasazení, škálování a správu kontejnerovaných aplikací. V současné době je funkční jádro LMCTFY součástí linuxové knihovny *libcontainer* a vývoj se tak přesunul tam.

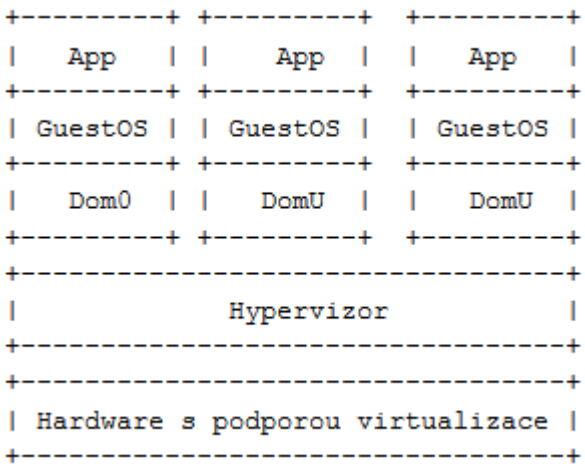
V témže roce vznikl projekt Docker, ze začátku jako interní projekt společnosti dotCloud. Jedná se o nyní široce využívaný systém pro správu kontejnerů. Dříve využíval knihovny LXC, které byly později nahrazeny knihovnou *libcontainer*. [1]

Iniciativa společnosti Microsoft vedla v roce 2016 ke vniku podpory pro kontejnerové technologie do operačního systému Microsoft Windows Server 2016. Toto řešení umožňuje spouštět Docker kontejnery nativně, bez nutnosti použití Linux VM. [3]

3.1.2 Plná virtualizace

Virtualizace je technika, která umožňuje v počítači zpřístupnit dostupné zdroje jiným způsobem, než v jakém fyzicky existují. Virtualizace celého operačního systému na sdíleném hardware se rozděluje na dva typy – hostovanou a nativní. V případě hostované virtualizace je na hostujícím operačním systému nainstalován software umožňující běh virtualizovaného operačního systému. Mezi řešení toho typu patří například VMware Workstation, VirtualBox a další.

Nativní virtualizace je řešena za pomocí virtualizační vrstvy, hypervizoru. Tento hypervizor je nainstalován přímo na hardware a umožňuje běh hostujících operačních systémů. Nativní virtualizaci umožňují řešení VMWare vSphere a další. [4] [5]



Obrázek 1 - Schéma plné nativní virtualizace [4]

3.1.2.1 Výhody

Mezi výhody virtualizace celého operačního systému patří možnost spuštění různých, nezávislých, operačních systémů nad sdíleným hardwarem. S tím je též spojena vyšší bezpečnost, protože pro napadení hostujícího systému je nutné provést útok skrze hostovaný operační systém. Podpora v OS není pro tento typ virtualizace nutná a je tak možné virtualizovat jakýkoli operační systém.

Virtualizační nástroje, zvláště ty, které jsou určené pro komerční využití, umožňují spravovat virtualizované stroje na mnoha fyzických počítačích. Umožňují také provádět

jejich úpravy a migrace včetně jejich připojených souborových systémů. Díky GUI je také možné je obsluhovat bez nutnosti hlubokých znalostí virtualizace a souvisejících služeb a aplikací. [5]

3.1.2.2 Nevýhody a omezení

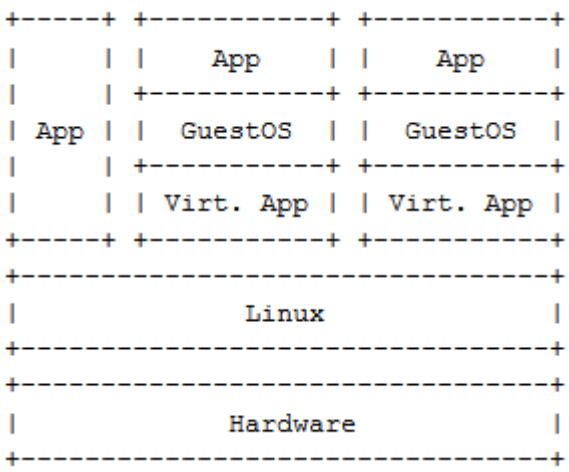
Velkou nevýhodou plně virtualizovaných systémů je jejich náročnost na výkon a místo především v případě hostované varianty. U nedostatečně výkonných počítačů, popřípadě jejich virtualizovaných ekvivalentů, se je často nevyplatí dále virtualizovat. Virtualizační nástroje totiž požadují alespoň jeden systém, popřípadě hypervizor, pro svůj provoz. Běh tohoto systému může znatelně omezit zdroje požadované pro hostované operační systémy. Toto omezení platí také pro úložiště dat, protože obrazy operačních systémů jsou často v jednotkách gigabajtů. [5]

Zprovoznění, instalace a bezpečné nastavení virtualizace vyžadují zkušenosti a pokročilé administrátorské znalosti s ohledem na nutnost nastavení firewallů, směrování atp.

3.1.3 Virtualizace na úrovni OS

Virtualizace na úrovni operačního systému, na rozdíl od virtualizace celého OS, je řešena takzvanými kontejnery, které sdílí s hostujícím operačním systémem jádro systému. V těchto kontejnerech je dostupný virtualizovaný operační systém. V něm běžící aplikace jsou izolovány od aplikací běžících mimo tento kontejner. [5] [6]

Mezi populární řešení tohoto typu virtualizace na operačním systému Linux patří například LXC, LXD, Docker a jejich orchestrační nadstavby jakými jsou Openshift a Kubernetes.



Obrázek 2 - Schéma virtualizace na úrovni OS [4]

3.1.3.1 Výhody

Mezi hlavní výhody virtualizace na úrovni OS patří rychlost nasazení kontejnerovaného systému, respektive aplikace, a velikost obrazů těchto kontejnerů. Proces instalace a zprovoznění kontejneru lze velmi dobře automatizovat, díky tomu je možné nepoužívané kontejnery mazat a šetřit tak místo na úložišti.

Virtualizace na úrovni OS s sebou také nese mnohem menší zátěž navíc, kterou provází běh plnohodnotného operačního systému. To umožňuje souběžnou činnost většího množství kontejnerů, něž kterého by bylo možné dosáhnout s plně virtualizovanými operačními systémy. [2] [3]

3.1.3.2 Nevýhody a omezeni

Jelikož sdílí kontejnerované OS jádro s hostujícím systémem, je nutné pro jejich běh mít alespoň jeden plnohodnotný operační systém. Z tohoto sdílení také plyne závislost na operačním systému. Tento operační systém je v případě většiny kontejnerových technologií limitován na OS Linux, není to již ale pravidlem.

Vzhledem k menšímu oddělení hostujícího a hostovaného operačního systému poskytuje kontejnerizace menší úroveň zabezpečení při napadení aplikace nebo systému v kontejneru. [2] [3]

3.1.4 LXC

LXC (Linux Containers) je nástroj pro správu systémových nebo aplikačních kontejnerů. Tyto kontejnery využívají virtualizaci na úrovni operačního systému Linux a pro jejich izolaci jsou využívány jak funkce jádra (jedná se například o kernel namespaces, cgroups a další.), tak na tom postavené aplikace jako je SELinux a Apparmor.

LXC je v současné době dostupné ve verzi 1.0 a 2.0 (LXD) a obě tyto verze jsou vedeny jako LTS. Podpora obou verzí je plánována na dobu pěti let (pro verzi 1.0 do 1. června 2019). I díky tomu, že je možné LXC provozovat na neupraveném jádru, mají všechny hlavní linuxové distribuce v oficiálních repositářích. [4]

Ačkoli je to možné, nedoporučuje se instalovat LXC i LXD na jeden operační systém. Důvodem je podobnost příkazů (LXC využívá příkazy začínající *lxc-*, zatímco LXD využívá jediný a to *lxc*) a také to, že LXD není jen nadstavba LXC, ale jeho další verze. Může se tak stát, že současné využívání LXC i LXD zmátne nejen uživatele, ale způsobí problém samotnému správci kontejnerů. Kvůli sdílení zdrojů jako jsou jmenné prostory mohou vniknout konflikty konfigurace bránící bezproblémovému provozu. [8]

V této práci je popisována funkcionalita a použití LXC, je možné dosáhnout stejného výsledku za využití LXD.

Kontejnery lze využívat ve dvou režimech: privilegovaný a neprivilegovaný. Privilegované kontejnery běží pod právy superuživatele na hostovaném operačním systému. Tato volba může přinášet potenciální bezpečnostní riziko v případě, že by útočník dokázal získat kontrolu nad kontejnerem. Na druhou stranu umožňují tato práva kontejnerům vytvářet přístupové body zařízení, takzvané nody, a připojovat souborové systémy.

Neprivilegované kontejnery využívají funkci jádra od verze 3.12, která se nazývá *cgroupy*. Ty umožňují spouštět kontejnery s právy uživatele, která jsou sice omezující, ale poskytují vyšší bezpečnost. Před spuštěním neprivilegovaného kontejneru je nutné nastavit tyto kontrolní skupiny, které mimo jiné umožňují mapovat ID superuživatele uvnitř kontejneru na uživatele hostujícího systému.

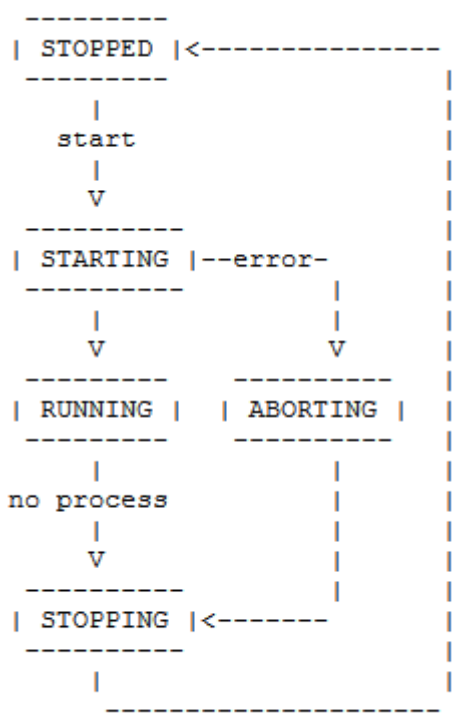
Cgroupy kromě izolace paměti umožňují též omezit systémové prostředky. Například omezit maximální využití procesoru, paměti atp. [2] [7]

Ačkoli jsou neprivilegované kontejnery podporovány již v linuxovém jádře, některé distribuce nejsou dostatečně kompatibilní a zpustit takový kontejner může být problém. Důvodem mohou být staré verze aplikací, špatně nastavený SELinux nebo parametry OS. [9] [10]

LXC kontejner je prakticky vlastní strom souborů uložený na hostujícím souborovém systému (například v adresáři **/var/lib/lxc/kontejner/rootfs/**). Tento strom reprezentuje souborový systém dostupný v kontejneru, a jeho konfiguraci. V případě neprivilegovaného kontejneru je vlastník i skupina podle nastavení *cgroup*, v opačném případě je vlastníkem uživatel *root*.

Konfigurace kontejneru obsahuje nastavení specifické pro danou architekturu a distribuci, cestu k *root* adresáři kontejneru, připojení složek z hostovaného systému, nastavení síťového rozhraní a další. Samotnou izolaci při spuštění zajistí skripty samotného LXC.

LXC kontejner má během svého životního cyklu několik stavů. Tyto stavy se odlišují hlavně tím, zda v kontejneru běží nějaký proces. Kontejner je celý ukončen spolu s ukončením jeho posledního procesu. Pokud během spouštění prvního procesu dojde k nějaké chybě, dojde k ukončení přes stav označený jako *aborting*, ukončování. [7]



Obrázek 3 - Životní cyklus LXC kontejneru [11]

Pro správu samotných kontejnerů slouží příkazy lxc-*. Jedná se o *lxc-create*, *lxc-start*, *lxc-attach*, *lxc-stop*, *lxc-destroy*, *lxc-info* a *lxc-ls*. V distribucích jsou často rozděleny mezi balíky *lxc* a *lxc-extra*. [7] [11]

3.1.4.1 Vytváření a mazání kontejnerů

Ačkoli je možné si kontejner vytvořit manuálně, není nutné tak činit, protože existuje mnoho předpřipravených obrazů různých distribucí i architektur. Obraz je archiv s nastavením a souborovým systémem kontejneru.

Pomocí příkazu *lxc-create* lze vytvářet samotné kontejnery z předpřipravených obrazů. Tyto obrazy bývají k dispozici nejen pro architekturu x86, resp. x86_64, ale často také pro arm a další.

Naopak příkaz *lxc-destroy* slouží ke smazání všech souborů kontejneru, včetně konfigurace, odstranění síťového rozhraní atp. [11]

3.1.4.2 Spouštění a zastavení kontejnerů

Příkaz *lxc-start* slouží ke spouštění systému uvnitř kontejneru. Jeho parametrem může být příkaz, který je proveden uvnitř kontejneru a ID tohoto procesu je 1. Bez parametru je spouštěn příkaz */sbin/init*, tedy stejně jako při klasické bezkontejnerové instalaci linuxového operačního systému.

Spuštěním tohoto příkazu dojde k připojení na konzoli kontejnerovaného systému. V případě, že nejsou nastaveny žádné přihlašovací údaje tak je efektivně znepřístupněna konzole, ze které byl příkaz spuštěn. Je tak výhodné příkaz používat s parametrem *-d*, který provede spuštění kontejneru bez jeho připojení. Do kontejneru se dá po jeho spuštění připojit příkazy *lxc-console*, či *lxc-attach*, popřípadě přes služby v něm spuštěné.

Ke spuštění kontejneru slouží též příkaz *lxc-execute*. Stejně jako při využití *lxc-start* je v parametru udaný příkaz spuštěn v kontejneru. Rozdíl je ale v tom, že před spuštěním tohoto příkazu je připojen adresář */proc*, vytvořen a spuštěn proces *lxc-init*. Tento proces má ID 1 a slouží jako podpora pro běh démonů v kontejneru. Proces příkazu má v tomto případě ID 2.

Tento příkaz lze využít i v případě, že kontejner ještě není vytvořený, ale je poskytnuta jeho konfigurace. V takovém případě je kontejner vytvořen a po ukončení posledního procesu odstraněn.

lxc-execute tedy na rozdíl od *lxc-start* slouží ke spouštění aplikací a příkazů.

Pro vynucené ukončení kontejneru existuje příkaz *lxc-stop*. Tento příkaz zastaví všechny procesy v kontejneru. Je tak vhodný pro případy, kdy systém v kontejneru neodpovídá, nelze ukončit klasicky příkazem *shutdown*, popřípadě není vůbec dostupný. [11]

3.1.4.3 Lxc-attach

V případě, že nejsou nastavené žádné přihlašovací údaje pro uživatele v kontejneru, ani není spuštěna žádná služba jako například SSH, slouží tento příkaz, podobně jako *lxc-console*, k připojení ke konzoli uvnitř kontejneru. *lxc-attach* provede připojení na konzoli uživatele *root* bez nutnosti zadání hesla. [11]

3.1.4.4 Lxc-cgroup

Tento příkaz umožňuje zobrazit a měnit parametry cgroup za běhu kontejneru. Je možné tak měnit počet dostupných procesorů pro kontejner a další. Uživatel musí zadat celé jméno subsystému cgroupy, který chce měnit. Není zde prováděna žádná kontrola platnosti příkazu a není tedy zaručeno, že bude požadavek na změnu proveden. [11]

3.1.4.5 Sledování stavu kontejnerů

Při větším množství kontejnerů může nastat problém se sledováním jejich stavů, názvů apod. Ke sledování kontejnerů tedy slouží několik příkazů, které vychází z klasických příkazů dostupných v Linuxu. Jedná se o *lxc-ls, lxc-ps, lxc-info* a *lxc-netstat*.

lxc-*ls* je nadstavba nad klasický příkaz *ls* a poskytuje seznam názvů kontejnerů. Jeho výstup lze použít pro získání informací o všech kontejnerech v dalších příkazech:

```
for i in $(lxc-ls -1); do
    lxc-info -n $i
done
```

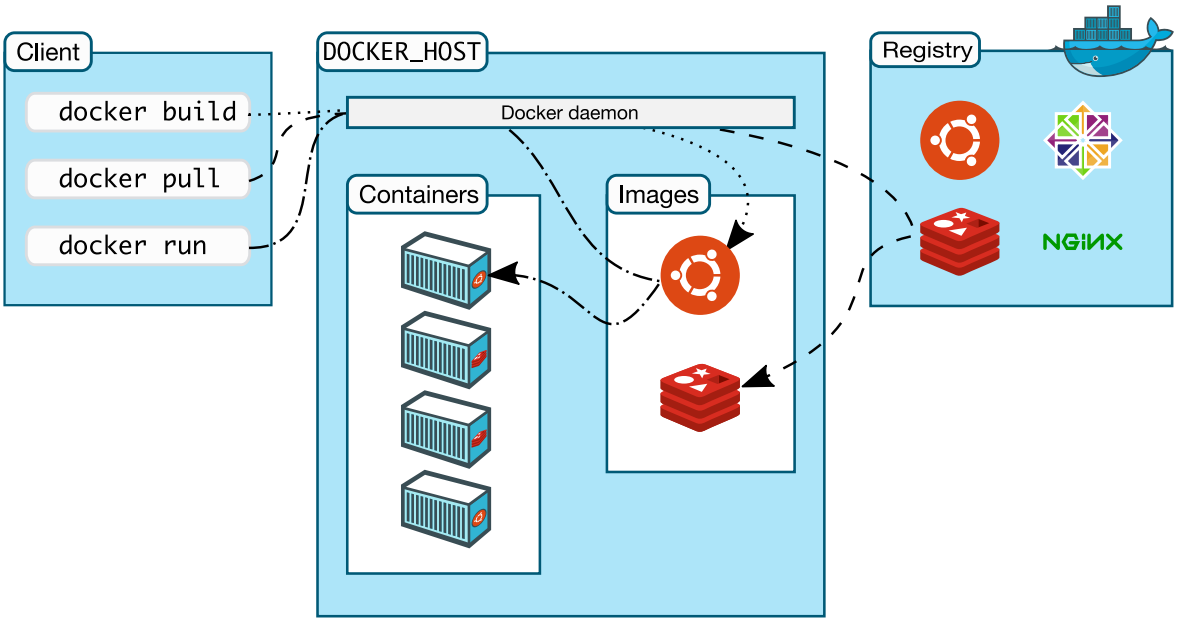
Nad linuxovým programem *ps* je postaven příkaz *lxc-ps* a slouží k výpisu procesů běžících v kontejneru. Pro získání informace o stavu kontejneru, tedy zda běží, či nikoli, slouží příkaz *lxc-info*. Pomocí příkazu *lxc-netstat* lze zjistit informace o síťovém provozu z a do kontejneru, stejně jako je poskytuje příkaz *netstat*. [11]

3.1.5 Docker

Docker je funkčně velmi podobný LXC. Mimo samotné izolace v kontejnerech nabízí ale větší možnosti automatizace a přináší lepší přenositelnost. Na rozdíl od LXC je tak vhodnější pro automatické vytváření kontejnerů, jejich přenos a spouštění na dalších hostujících systémech. To je dáno také tím, že zatímco cíl LXC je spíše vytváření a běh perzistentních virtuálních strojů, Docker je určen hlavně jako platforma pro správu jednotlivých izolovaných aplikací. [12] [13]

Docker staví na stejných technologiích jako LXC, místo kontejnerů distribucí ale pracuje primárně s kontejnery aplikací. Ty jsou dostupné jako obrazy v Docker repositářích a je tak možné rychle získat a zprovoznit prostředí pro běh aplikací, popřípadě

aplikace samotné. Na rozdíl od samotného LXC je možné provozovat Docker kontejnery i v prostředí operačního systému Microsoft Windows. [3]



Obrázek 4 - Architektura Docker technologie [12]

Díky poskytovanému API vniklo mnoho aplikací, který Docker využívají a které automatizují vytváření kontejnerů, jejich běh a případný zánik. To vede k tomu, že na rozdíl od LXC kontejnerů Docker kontejnery často existují jen po dobu jejich běhu.

To přineslo zcela jiný přístup k práci s těmito kontejnery a aplikacemi v nich běžícími. Veškerá trvalá data se ukládají mimo kontejner a buď jsou dostupné při vytváření kontejneru, přístupná jako připojené soubory a adresáře, popřípadě přes síťová rozhraní. Docker kontejnery ale neřeší vzájemnou závislost při spouštění, což může vést k tomu, že se kontejner s databázovou aplikací spustí až po kontejneru s aplikací, která jí vyžaduje. Tento problém je nutné řešit ručně, na úrovni aplikace nebo je nutné využít orchestračních nadstavbách jako je Kubernetes a OpenShift. [2] [6]

Na rozdíl od LXC má docker jediný příkaz, *docker*, kterým se provádí veškerá správa kontejnerů skrze parametry. Jedná se například o *docker build*, *docker run* a *docker container*. Samotné kontejnery se nastavují přes konfigurační soubor, který obsahuje kombinaci parametrů pro službu Docker a příkazů, které jsou spuštěny při vytváření kontejneru. Tento soubor se jmenuje *Dockerfile*.

Dále je možné nastavit chování Docker služby vzhledem ke kontejneru. K tomuto účelu slouží soubor *docker-compose.yml*. Tento soubor umožňuje spouštět kontejner jako službu se zadaným chováním. Je možné zde omezit paměťové a výpočetní prostředky, mapovat porty kontejneru na porty hostujícího systému, stanovit škálování a další. [12]

Příkladem může být následující konfigurace:

```
version: "3" #verze konfigurace
services:
  web:
    image: username/repo:tag #název a tag kontejneru
    deploy:
      replicas: 5 # nastavení počtu spuštěných instancí
      resources:
        limits: # omezení výpočetních a paměťových
zdrojů
          cpus: "1.0"
          memory: 300M
      restart_policy:
        condition: on-failure
    ports: # nastavení mapování portů mezi kontejnerem a
hostovaným systémem
      - "80:80"
    networks: # nastavení sítě, která na kterou bude
kontejner připojen
      - webnet
networks: # dodatečné nastavení jednotlivých sítí
  webnet:
```

3.1.5.1 Docker build

Příkaz *docker build* umožňuje sestavit vlastní kontejner za využití již existujících kontejnerů, které jsou dostupné v Docker repositářích. Základem vytváření kontejnerů je soubor *Dockerfile*, který obsahuje veškerou konfiguraci kontejneru. V rámci volání tohoto příkazu je možné zadat tag, který slouží k pojmenování nového repositáře kontejnerů.

Docker build -t container . vytvoří kontejner, který je pak dostupný v lokálním docker repositáři *container* s tagem *latest* a unikátním ID. Tečka na konci odkazuje na umístění *Dockerfile* a případných dalších souborů. Díky tomu je možné vytvářet různé verze jednoho typu kontejneru, například s různými verzemi použitého softwaru. Jeden kontejner může mít více tagů a může být ve více repositářích. [12]

Parametr *-t* může obsahovat název repositáře i tag, a to zápisem *username/repository:tag*. Výsledek pak může být následující:

```
$ docker build -t container .
$ docker build -t xkrum003/container:1.0.0 .
$ docker image ls

REPOSITORY            TAG                 IMAGE ID

container             latest              326387cea398
xkrum003/container    1.0.0               a5a4c2248b4a
```

3.1.5.2 Docker run

Příkaz *docker run* slouží k ručnímu spouštění kontejnerů, respektive kontejnerovaných služeb. Povinným parametrem je jméno, nebo tag kontejneru.

```
$ docker run container-name
```

Stejně jako LXC kontejnery, lze Docker kontejnery spouštět parametrem s připojením na konzoli, nebo parametrem *-d* provést spuštění na pozadí. Pokud není kontejner k dispozici lokálně, dojde automaticky k pokusu o stažení kontejneru z repositáře.

Během spuštění lze připojit port kontejneru na port v hostovaném operačním systému, a tak ho zpřístupnit z vnější sítě. Docker tak řeší směrování portů na rozdíl od LXC sám a není nutné pak provádět ruční změny na úrovni firewallu. Číslo portu lze doplnit o omezení na protokol TCP, nebo UDP. Těchto pravidel lze psát více opakováním parametru *-p*. Následující příkaz tedy spustí na pozadí kontejner a kontejnerový port 80 bude dostupný na hostujícím systému na portu 8080 pro protokol TCP.

```
$ docker run -d -p 8080:80/tcp container-name
```

Pokud není parametrem *–name* přidáno jméno kontejneru, které musí být unikátní pro každý vytvořený kontejner, je mu vytvořeno náhodné. Toto jméno je poté použito jako parametr při ovládání kontejneru. [12]

3.1.5.3 Docker container

Skupina příkazů v rámci *docker container* slouží k monitorování, mazání a vypínání kontejnerů.

d*ocker container ls* poskytuje informace o běžících kontejnerech, jejich jména, unikátní id, spuštěné příkazy v nich, porty a další informace. Parametrem *-a* lze získat informace o všech instalovaných kontejnerech.

Pro vypnutí kontejneru je určen příkaz *docker container stop* a *docker container kill*. Tyto příkazy provedou klasické vypnutí, respektive vynucené zastavení, kontejneru, jehož id je uvedeno jako parametr.

Smazání kontejneru lze provést příkazem *docker container rm*. V takovém případě dojde ke smazání kontejnerových souborů ze souborového systému hostujícího OS. Takový kontejner může být ale dále k dispozici pro spuštění přes repositář. [12]

3.1.5.4 Dockerfile

Dockerfile soubor obsahuje jak příkazy, které jsou provedeny během sestavení kontejneru, tak instrukce pro Docker při jeho spuštění. Tyto příkazy umožnují rozšíření existující šablony pro kontejner, spuštění jakéhokoli příkazu v kontejneru, zkopírování souborů z hostujícího systému do kontejneru a další. [12]

Příkaz	Popis
FROM	Využije již existující šablonu kontejneru z repositáře zadanou jako parametr
WORKDIR	Změní pracovní adresář v kontejneru na zadaný v parametru
ADD	Zkopíruje obsah adresáře z hostujícího OS z prvním parametru do adresáře v kontejneru uvedeném v druhém parametru
RUN	Spuštění jakéhokoli příkazu v kontejneru uvedeného za tímto klíčovým slovem
EXPOSE	Povolí tok paketů do kontejneru na portu uvedeném v parametru
ENV	Nastavení proměnné prostředí v kontejneru z parametrů po dobu sestavování.
CMD	Příkaz v parametrech je spuštěn během spuštění kontejneru

Tabulka 1 - Dockerfile příkazy [12]

3.2 Webové služby

Webová služba je řešení v síti Internet, které umožňuje komunikaci mezi dvěma koncovými body předepsaným, standardizovaným způsobem a je nezávislá na použitém softwaru koncových zařízení. Tato komunikace může probíhat mezi klienty ve vztahu klient-server i klient-klient. Webové služby lze rozdělit podle účelu a využitých komunikačních protokolech na webový server, mailový server, databázový server a další.

3.2.1 Webový server

Webovým serverem bývá označován software, fyzický či virtualizovaný počítač dostupný na síti Internet. Ten je na odpovědný za odpovídání na požadavky přicházející na definovaných protokolech od klientů. V této práci bude jako webový server označena aplikace poskytující služby na protokolech HTTP a HTTPS.

Předepsaný port pro protokol HTTP je TCP/80 a pro protokol HTTP TCP/443, webový server ale může poskytovat služby na jakémkoli portu TCP. [14]

Hlavním účelem webového serveru je poskytovat přístup k webovým stránkám. Odpovědí serveru ale může být i soubor, nebo proud dat (stream) – jako je vysílání videa, či rádia.

Protokol HTTP byl poprvé definován v roce 1991 a verze HTTP/1.0 byla publikována v roce 1996. V současné době se využívá hlavně verze 1.1 z roku 1999. Na portu 80 lze též poskytovat protokol HTTP/2, jehož RFC bylo vydáno v roce 2015. Žádný z webových prohlížečů ale nebude nešifrovanou verzi tohoto protokolu podporovat. [15]

Protokol HTTP je nešifrovaný, bezstavový a komunikace probíhá v případě HTTP/1 v čistém textu. Komunikace na protokolu HTTP/2 je binární. Přenášená data je tak jednoduché odposlechnout, pozměnit, či podvrhnout z pozice MitM, aniž by to klient, či server, zjistil.

Z tohoto důvodu byl zaveden protokol HTTPS, jedná se o HTTP přenášený po síti přes protokol SSL/TLS. HTTPS při správné implementaci zaručuje zabezpečení dat šifrováním a zamezuje jejich pozměnění. Přenášená data jsou tak odolnější před odposlechem a dalšími útoky. Díky využité asymetrické kryptografii při navazování spojení lze též zajistit kontrolu autenticity klienta i serveru, k čemuž slouží soukromé klíče a veřejné klíče ve formě certifikátů. Tyto certifikáty lze též použít i pro zabezpečení dalších služeb poskytovaných přes protokol SSL/TLS.

Cenou za zvýšenou bezpečnost je zvýšená výpočetní náročnost z důvodu šifrování a dešifrování. Vzhledem k výkonům současných klientských zařízení i serverů je však tato zvýšená náročnost akceptovatelná. Výjimkou mohou být například domácí routery a IoT zařízení v zabezpečené síti.

Z tohoto důvodu se v čím dál větší míře nasazuje HTTPS a v mnoha případech webové stránky nejsou již na HTTP dostupné, a to i z důvodu podpory HTTP/2 pouze přes SSL/TLS v prohlížečích. [15] [16]

Mezi nejpoužívanější aplikace poskytující služby webového serveru patří Apache, Nginx a Microsoft-IIS. [17]

3.2.2 Mailový server

Mailovým, či emailovým, serverem je označován server, respektive aplikace na něm běžící. Slouží jako MTA a přenáší elektronickou poštu pomocí protokolu SMTP, či SMTPS. Tato komunikace probíhá buď v režimu klient-server pod označením MDA, nebo server-server jako MTA. Tedy přenos zpráv od uživatele k jeho emailovému serveru, nebo mezi mailovými servery.

Protokol SMTP/S slouží k přenosu emailové pošty k mailovému serveru adresáta přes jeden a více mailových serverů.

Mailový server může sloužit též jako MDA a v tom případě se stará o uložení zpráv do schránky adresáta. V takovém případě komunikace probíhá přes protokoly POP3 a IMAP, respektive POP3S a IMAPS. Komunikace s MDA probíhá pouze v režimu klient-server.

Protokoly IMAP/S a POP3/S slouží klientům k získání přístupu a správě jejich elektronické pošty. Protokol IMAP/S pracuje s poštou umístěnou na mailovém serveru, zatímco protokol POP3/S slouží ke stáhnutí poštovních zpráv ze serveru na zařízení klienta.

Komunikace pomocí protokolů SMTP, POP3 i SMTP probíhá v nešifrované podobě v čistém textu. Trpí tak stejnými bezpečnostními problémy jako protokol HTTP. Z tohoto důvodu se v čím dál větší míře využívá komunikace přes protokol SSL/TLS, který umožňuje zabezpečit čtení a manipulaci s přenášenými daty. Ze své povahy také může sloužit jako způsob boje proti spamu, protože je možné kontrolovat autenticitu serveru skrze certifikáty.

Pro protokoly využívané v přenosu elektronické pošty jsou vyhrazené porty (viz. Tabulka 2). V případě využití základních portů je buď celá komunikace nešifrovaná, nebo se vytváří šifrovaný kanál až od příkazu STARTLS. Jelikož je možné tuto část komunikace ovlivnit z pozice MitM, je bezpečnější využití portů, na kterých se očekává šifrovaná komunikace od začátku. [18]

Protokol / Port	Nešifrovaný / STARTTLS	SSL/TLS
SMTP	25	465, 587 (doporučeno)
POP3	110	995
IMAP	143	993

Tabulka 2 - Seznam vyhrazených portů pro protokoly SMTP/S, POP3/S a IMAP/S [20]

3.2.3 Databázový server

Databázovým serverem můžeme označit server, respektive aplikaci na něm běžící, která zpřístupňuje úložiště dat – databázi. Tato data mohou být strukturovaná i nestrukturovaná. Přístup je umožněn přes specifikované rozhraní nebo port.

Na rozdíl od webových serverů pro ně neexistují jednoznačně přiřazené porty a protokoly. Je to dané tím, že databázové servery nejsou ze své povahy většinou veřejně dostupné. Různé databázové aplikace tak využívají různé protokoly a porty, popřípadě jsou dostupné jen přes unixový *socket.* Socket je koncový bod pro umožňující komunikaci mezi procesy v rámci jednoho operačního systému. [19] [21]

Název databázové aplikace	Výchozí port pro komunikaci přes TCP/IP
MySQL	3306
PostgreSQL	5432
Microsoft SQL Server	1443
Firebird	3050

Tabulka 3 - Příklady výchozích portů pro některé databázové servery

Díky historickému vývoji jsou v současné době nejrozšířenější databáze relační, které mají pevnou tabulkovou strukturu na úrovni databáze a jsou označované jako SQL. Takové jsou všechny databáze zmíněné v tabulce. [21]

Existují se též NoSQL databáze. Jedná se o dokumentové, grafové, sloupcové, úložiště klíč-hodnota a další. Vzhledem ke své struktuře a parametrům se často hodí při řešení specifických problémů jako je třeba vyhledávání textu, „kešování“ a další.

3.3 Měření výkonu

Měřením výkonu je v této práci pravidelný sběr dat o vybraných dostupných prostředcích. Jmenovitě se jedná o místo na souborovém systému, paměti a procesorový čas. Tato data mohou být ukládána na hostujícím systému, popřípadě získávána a agregována vzdáleně.

3.3.1 Collectd

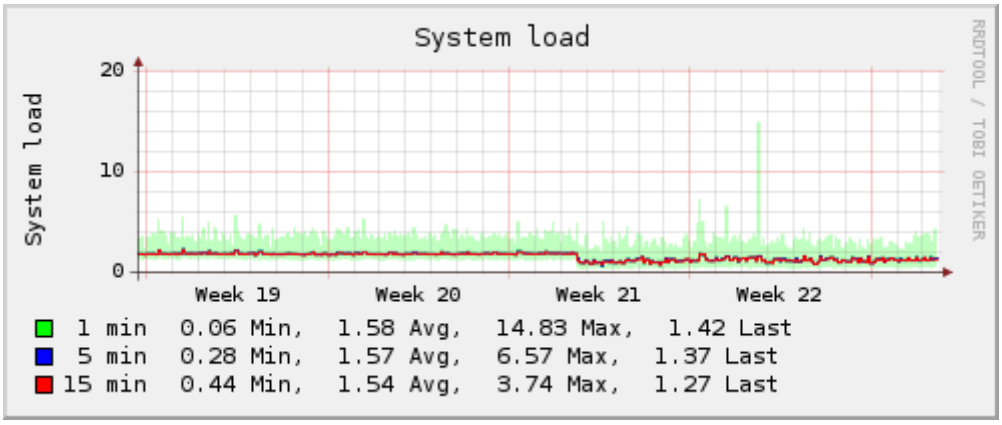
Collectd data pravidelně získává, agreguje a ukládá. Získaná data lze ukládat lokálně, například do databáze RRD pomocí RRDTool pluginu, z níž je pak možné číst a prezentovat data v aplikacích jako je Cacti. K dispozici jsou ale také pluginy pro ukládání do CSV, PostgreSQL a dalších formátů a aplikací.

Veškeré nastavení Collectd probíhá přes konfigurační soubory, které se nacházejí obvykle v adresáři **/etc/collectd/conf.d/**. Zde je jak nastavení samotného démona, tak instalovaných pluginů.

Samotná data jsou ukládána buď ve formě naměřené hodnoty, změny hodnot za čas, či naměřené hodnoty za čas. [22]

3.3.2 RRDTool

RRDTool je nástroj umožňující zápis, čtení a správu dat RRD databáze. Veškeré operace se provádí přes příkaz *rrdtool* a jeho parametry. Data je možné exportovat v agregované formě ve formě textového výpisu, XML a grafů. Pro tyto operace slouží operace *fetch* a *graph*, respektive *graphv*. [23]



Obrázek 5 - Grafový výpis nástroje RRDTool [22]

3.3.2.1 rrdtool fetch

Příkaz *rrdtool fetch* umožňuje získat data z RRD databáze ve formě textového výstupu. Tento příkaz je také interně využíván při generování výpisu ve formě grafu. Příkaz umožňuje omezit vypsaná data intervalem a agregovat je. Ve výchozím chování obsahuje výpis právě naměřené hodnoty, pro výpis je ale možné získat jejich průměr, minimální, maximální a poslední hodnoty v rámci časového intervalu. Tento interval musí být násobkem intervalu naměřených dat. [23]

Následujícím způsobem je možné získat data využití paměti za poslední minutu měřená po 10 vteřinách:

```
$ rrdtool fetch memory-used.rrd AVERAGE -s -300 -a


1520171730: 2.3536680960e+08
1520171740: 2.3542251520e+08
1520171750: 2.3535042560e+08
1520171760: 2.3535697920e+08
1520171770: 2.3540244480e+08
1520171780: 2.3541063680e+08
```

Hodnota 2.3536680960e+08 odpovídá 235366809.60 B, což je 224 MiB.

3.4 Bezpečnost

Mezi důležité důvody využití kontejnerizace a využívání virtualizovaných operačních systémů nepochybně patří vyšší bezpečnost a omezení škod způsobených napadením jednotlivých služeb. Kontejnery totiž mohou v případě napadení jedné služby zamezit škodám na hostovaném systému, popřípadě dalších nezávislých službách v jiných kontejnerech.

V případě neprivilegovaných kontejnerů jsou také omezeny škody v případě úspěšného úniku z kontejneru útočníkem. V takovém případě nezískává útočník neomezená práva na hostovaném systému. [6]

3.4.1 RCE

Vzdálené spuštění kódu, či spuštění libovolného kódu, je typ útoku, při kterém je útočník schopen provést na cílovém systému operace, které by za normálních okolností neměl mít možnost uskutečnit. Jedná se často o softwarové chyby, při kterých není dostatečně kontrolován vstup od uživatele. Zranitelnost RCE umožňuje spustit další typy útoků a jejich následkem může být vynesení informací, smazání dat, jejich zašifrování atp.

V rámci kontejneru může tyto následky minimalizovat například správně nastavený SELinux. Využití kontejneru umožňuje omezit útočníkovu oblast působnosti a zamezit škodám na dalších službách, popřípadě hostovanému systému. [24]

3.4.2 Eskalace oprávnění

Eskalace oprávnění často navazuje na útok typu RCE. Často je cílen na uživatele *root*, který nemá omezená práva. V případě úspěšného převzetí práv uživatele *root* v kontejnerovaném systému je útočník schopen způsobit škody. Pro napadení hostovaného systému je ale nucen provést další útoky na samotný kontejner a jádro systému.

V případě neprivilegovaných kontejnerů je i v případě úspěšného prolomení kontejnerizace útočník omezen jen na uživatele daného kontejneru. Značně se tak snižuje, v případě úspěšné a včasné detekce útoku, šance na úplné převzetí cílového systému a útok na ostatní služby. [19]

3.4.3 Čtení paměti aplikací

Tento typ útoku je velmi nebezpečný, protože může útočníkovi poskytnout soukromá data jako jsou hesla, soukromé klíče a další informace nutné k proniknutí do systému. Kontejner chrání ostatní data v případě, že je k útoku využívána chyba v implementaci, nebo návrhu softwarové aplikace. Příkladem může být chyba Heartbleed v sadě knihoven OpenSSL. Ta umožňovala nezjistitelně číst data aplikací z paměti. Díky izolaci paměti je možné omezit množství potenciálně dostupných informací. [25]

Pokud se ale jedná o útoky přímo na hardware, jakým je Meltdown a Spectre, nejsou kontejnery LXC ani Docker schopné poskytovat ochranu. Vzhledem k tomu, že je ale útočník nucen získaná data nějak exfiltrovat ze serveru, poskytují kontejnery další možnosti, jak takový tok dat zjistit a zamezit mu. [26]

3.4.4 DoS

DoS a DDoS útoky jsou velice populární, a to hlavně díky své jednoduchosti a obtížné obraně. V případě, že je DoS veden za účelem vytížení systémových zdrojů na cílovém serveru, poskytují kontejnery nástroje na omezení škod díky funkcím *cgroup*. Je tak možné zamezit úplnému vytížení hostovaného serveru, a s tím omezení ostatních služeb, pokud je útok veden na software v kontejneru. Je také možné přesměrovávat provoz vedený na kontejner, popřípadě kontejner vypnout, bez toho, aby byly omezeny služby poskytované skrze ostatní kontejnery, respektive samotný hostovaný server. [27]

Často je ale veden tzv. hloupý DDoS útok s cílem vytížit internetové připojení cílového serveru. V takovém případě je nutné provádět opatření již na cestě k serveru, na síťových prvcích.

3.5 Firewall

Firewall je aktivní síťový prvek, respektive aplikace, která plní roli firewallu. Slouží jako kontrolní bod aplikující nastavená pravidla na síťový provoz. Moderní firewally umožňují kontrolu toku dat, jejich omezení a monitoring a jsou tak důležitou součástí správy sítě a její zabezpečení. [19]

3.5.1 UFW

Uncomplicated firewall je výchozí nástroj k nastavení pravidel pro kontrolu toku dat na síťových rozhraních v linuxové distribuci Ubuntu. Jedná se o terminálovou aplikaci, která usnadňuje tvorbu a změnu *iptables* pravidel. Ve výchozím nastavení je vypnutý a seznam *iptables* pravidel je prázdný, příchozí i odchozí datový tok tak není jakkoli omezován ani měněn.

UFW nabízí mimo jiné jednoduché příkazy, které umožňují zapnutí a vypnutí firewallu, povolení služby, či portu atp.

Uncomplicated firewall bohužel neobsahuje možnost nastavení přesměrování portů, která je nutná pro zpřístupnění služeb běžících v LXC kontejnerech. Tato pravidla je nutná nastavit přímo přes příkazy aplikace *iptables*. [28]

3.5.1.1 Povolení a zakázání portů

Pro povolení a zakázání příchozích paketů na vnější rozhraní slouží příkazy *ufw allow* a *ufw deny*. Tyto příkazy přepisují výchozí chování, které je definované v souboru **/etc/default/ufw**. Parametrem může být buď název šablony instalované aplikace, název služby podle souboru **/etc/services,** nebo číslo portu s volitelně udaným protokolem TCP, nebo UDP. [28]

Všechny následující příkazy způsobí po aplikaci změn povolení příchozích TCP paketů na portu 22 (SSH):

```
$ sudo ufw allow `Open SSH`
$ sudo ufw allow ssh
$ sudo ufw allow 80/tcp
```

Toto nastavení je poté uloženo jako pravidlo programu *iptables.* Následující výpis je zkrácen. [28]

```
$ sudo iptables -L

Chain ufw-user-input (1 references)
ACCEPT   tcp   --   anywhere   anywhere      tcp dpt:ssh
```

4 Praktická část

Instalace, konfigurace a testování jsou prováděny na plně virtualizovaném operačním systému Linux. Jako distribuce je zvoleno Ubuntu z důvodu jeho velké rozšířenosti, a také protože nabízí nejlepší podporu pro běh kontejnerů LXC a Docker. [29]

Systémy plně virtualizované službou KVM, hostované firmou WEDOS Internet, a.s. a mají stejné parametry. Jedná se o 15 GB SSD, 2 GB DDR3 dedikované RAM a 1 vlákno procesoru Xeon 1.7 GHz. Tato konfigurace je dostatečně výkonná pro testované služby a současně je možnost vzájemného ovlivnění výkonu zátěží jednotlivých virtualizovaných strojů považována za zanedbatelnou.

Hostující operační systémy jsou dostupné přes protokoly IPv6 i IPv4. Důvodem pro využití IPv4 je nedostatečná podpora a omezení na straně použitých kontejnerů v případě nedostupnosti IPv4 na hostovaném operačním systému. S ohledem na sledování výkonu je jakýkoli vnější provoz nežádoucí, a tak byl blokován na úrovni firewallu. Veškeré služby budou nastaveny tak, aby byly dostupné na svých výchozích portech z internetu na obou protokolech.

Jednotlivé VPS mají jako *hostname* nastavené *dp-bare*, *dp-lxc* a *dp-docker* podle toho, zda a jakou kontejnerovou technologii využívají. Pod těmito adresami je na ně později přistupováno během instalace, nastavení služeb a testování výkonu. Jako doména 2. řádu je zvolena adresa *duke-hq.net* vlastněná autorem této práce.

Instalace a správa Docker kontejnerů je částečně využitelná i administrátory Windows serverů. To je umožněno aplikací Docker for Windows, která umožňuje běh linuxových aplikací v kontejnerech v prostředí OS Windows. [3]

Demonstrační systém bude využívat Ubuntu 16.04 LTS a zabezpečované služby bude poskytovat následující software:

Služba	**Software**	**Verze**	**Kontejnerizace**
SSH, SFTP	OpenSSH	7.2	Žádná
HTTP, HTTPS	Apache	2.4.18	Žádná, LXC, Docker
SMTP, SMTPS	Postfix	3.1.0	Žádná, LXC, Docker
Databáze	MySQL	5.7.21	Žádná, LXC
Sběr statistik	Collectd	5.5.1	Žádná

Tabulka 4 - Seznam instalovaných služeb a aplikací

S ohledem na možnou odlišnost verzí i jejich dostupností v různých distribucích není zaručeno, že konfigurační soubory dostupné v rámci přílohy i této práce budou plně přenositelné. V distribucích založených na Ubuntu využívající balíčkovací software *apt* je příkaz na zjištění aktuální instalované verze následující.

```
$ sudo apt show <package>
```

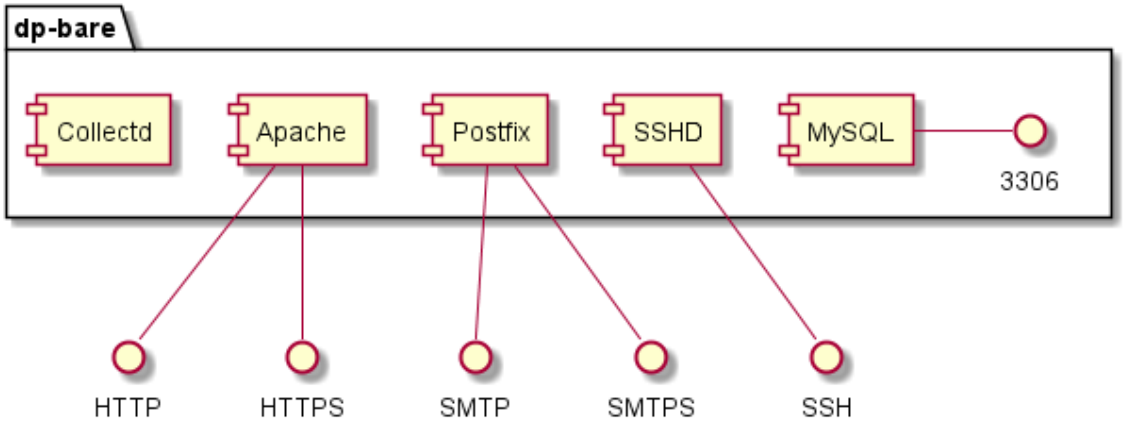
Databázový server MySQL je vyjmut z Docker kontejnerizace z důvodu závislosti ostatních služeb na tomto serveru, kterou samotný Docker není schopný uspokojivě vyřešit. Docker, stejně jako LXC, není schopný řešit postupné startování kontejnerů, ani jejich restarty z důvodu jejich závislosti. Tyto požadavky lze řešit až v rámci orchestračních nadstaveb jako je Kubernetes a OpenShift. Přesto nepatří databázové servery k často kontejnerovaným službám i díky možným problémům s daty v případě automatického restartu, který tak nemusí proběhnout v pořádku.

Služba SSH a SFTP poskytovaná aplikací OpenSSH není součástí kontejnerů a slouží pouze ke správě a nastavení ostatních služeb. V případě Docker kontejnerů probíhá jejich konfigurace a nastavení předem a jakékoli změny v běžícím kontejneru jsou považovány za dočasné. Pro přístup do kontejnerů LXC není nutné využívat službu SSH. Místo ní se lze připojit do shellu, což je uživatelské rozhraní pro přístup ke službám OS, v kontejneru pomocí příkazu *lxc-attach*.

Sběr statistických dat bude probíhat aplikací Collectd, která bude data ukládat mimo kontejnery do RRD databází. Neexistuje tedy důvod spouštět tuto službu v kontejneru s ohledem na to, že výkon jednotlivých kontejnerů není v této práci důležitý.

4.1 Příprava serveru a služeb

Některé služby a nastavení jsou společné pro všechny testované konfigurace. Jedná se o službu SSH a aplikaci Collectd. Kontejnerované služby využívají podobnou konfiguraci, jakou by měla služba běžící mimo kontejner. Rozdíly obvykle plynou z omezení a vlastností samotných kontejnerů. Velký rozdíl je v nastavení firewallu, a tedy i zpřístupnění služeb přes porty, kdy je nutné pro LXC kontejnery vytvářet pravidla pro přesměrování portů. Docker kontejnery je možné spustit tak, že naslouchají spojením na portech hostujícího systému.



Obrázek 6 - Schéma instalovaných služeb bez kontejnerů

4.1.1 SSH – OpenSSH

Protokol SSH, poskytovaný aplikací OpenSSH, bude využíván pro veškerou práci na serveru a nastavení jeho služeb. V základní instalaci je dostupný pouze uživatel root, který má neomezená práva. S ohledem na to je třeba upravit konfiguraci SSH démona a co

nejdříve vytvořit klasického uživatele. Vytvoření uživatele a jeho povolení změn přes *sudo* lze docílit následujícími příkazy:

```
# adduser <username>
# usermod -a -G sudo <username>
```

Od tohoto momentu je možné se přihlásit na server i pod tímto uživatelem. Z bezpečnostních důvodů je od této chvíle vhodné zamezit přístupu přes SSH pro uživatele root. V rámci této práce bude přístup pro uživatele možný pouze přes tzv. klíče, které je nutné vytvořit předem a poté nahrát veřejný klíč na server. Soukromý klíč musí zůstat na klientovi. Výchozí konfigurace OpenSSH využívá pro veřejné klíče soubor **.ssh/authorized_keys** v adresáři uživatele. Nahrát veřejný klíč lze například následujícím způsobem:

```
mkdir ~/.ssh
echo "public key" > ~/.ssh/authorized_keys
chmod -R 700 ~/.ssh
```

Popřípadě přímo z klienta přes linuxovou aplikaci *ssh-copy-id*:

```
ssh-copy-id username@remote_host
```

Výchozí nastavení SSH je po instalaci připravené pro běžné používání, ale je vhodné zvýšit zabezpečení této služby změnou některých parametrů. Konfigurace OpenSSH se obvykle nachází v souboru **/etc/ssh/sshd_config.** Změnou následujících parametrů je zamezeno přímému přihlášení uživateli *root* a zkrácena doba pro přihlášení, než dojde k vynuceném odpojení. Zkrácení doby pro přihlášení zvyšuje obranu proti útokům typu DoS. Změněné hodnoty oproti výchozí konfiguraci jsou následující:

```
LoginGraceTime 1m
PermitRootLogin no
```

Parametr	**Popis**
LoginGraceTime	Čas, za kterou se uživatel musí přihlásit
PermitRootLogin	Zda je povolené přihlášení přes SSH pro uživatele root

Tabulka 5 – Vybrané konfigurační parametry SSH [30]

4.1.2 Collectd

V základní instalaci Ubuntu 16.04 LTS není Collectd nainstalován. Instalaci odpovídajícího balíku *collectd* lze provést přes *apt-get.*

Tento balík obsahuje samotnou aplikaci a základní pluginy. *Collectd-utils* poskytuje kromě dalších pluginů také aplikaci *rrdtool*, která slouží pro čtení dat z RRD databáze, do které jsou sbíraná data ukládána. Ve výchozím nastavení jsou data aktualizována každých 10 vteřin a data starší 3 hodin jsou ukládána po jedné minutě, což je pro potřeby následné analýzy výkonu dostatečné.

Ve výchozím nastavení sbírá *collectd* data z mnoha zdrojů. Pro potřeby této práce je nutné povolit jen některé. Jedná se o pluginy *load*, pro měření zátěže procesoru, *df* a *memory* pro měření využití HDD a RAM. Plugin *syslog* je využit pro záznam do systémového logu a *rrdtool* je plugin, který sbíraná data ukládá do RRD databáze. Výsledný konfigurační soubor **/etc/collectd/collectd.conf** obsahuje následující:

```
Hostname "dp-bare"
FQDNLookup true

LoadPlugin syslog

<Plugin syslog>
     LogLevel info
</Plugin>

LoadPlugin df
LoadPlugin load
LoadPlugin memory
LoadPlugin rrdtool
```

Při sledování zaplnění disků bude pro potřeby analýzy důležitý pouze disk připojený jako kořenový adresář. Ve výchozím nastavení plugin *df* sleduje všechny připojené disky, což je zbytečné také protože kontejnery LXC i Docker vytvářejí při svém běhu virtuální připojené disky.

```
<Plugin df>
     MountPoint "/"
     IgnoreSelected false
</Plugin>
```

```
<Plugin rrdtool>
     DataDir "/var/lib/collectd/rrd"
</Plugin>

<Include "/etc/collectd/collectd.conf.d">
     Filter "*.conf"
</Include>
```

Název pluginu	**Popis**
Syslog	Provádí zápis do systémového logu
Cpu	Sledování času procesoru v jeho jednotlivých stavech
Df	Sledování zaplnění připojených souborových systémů
Disk	Sledování vytížení disků z hlediska čtení a zápisu
Load	Sledování vytížení procesorového času
Memory	Sledování využití paměti
Processes	Sledování počtu běžících procesů a jejich stavů
Rrdtool	Provádí zápis získaných dat do RRD databáze

Tabulka 6 – Funkcionalita Vybraných pluginů Collectd [31]

Tento konfigurační soubor je shodný pro všechny varianty instalace, jediná rozdílná hodnota je *Hostname*, která odpovídá názvu serveru. Veškeré pluginy bez konfiguračních sekcí využívají výchozí hodnoty.

4.2 Příprava kontejnerů

Kontejnery LXC a Docker mají ze své podstaty a využití různé požadavky na nastavení, které je nutné splnit pro úspěšně spuštění kontejneru a služeb v něm. Tato práce pracuje s verzemi kontejnerů vypsaných v následující tabulce.

Název typu kontejneru	**Instalovaná verze**
LXC	2.0.8
Docker	1.5

Tabulka 7 - Instalované verze kontejnerovacích aplikací

4.2.1 LXC

Před spuštěním jakéhokoli neprivilegovaného kontejneru je nutné nastavit korektní mapování root uživatele v kontejneru na skutečného uživatele, pod kterým běží instance LXC kontejneru na hostujícím operačním systému.

Z bezpečnostních důvodů je doporučené vytvořit dedikovaného uživatele pro běh LXC kontejneru příkazem *adduser* či *useradd*. Demonstrační instalace využívá jediného uživatele z důvodu zjednodušení konfigurace, ta by byla pro každého uživatele podobná. V prvé řadě je nutné zjistit povolený rozsah id podskupin a poduživatelů, které může daný uživatel „obsluhovat“. To je možné ze souborů */etc/subgid* a */etc/subuid*:

```
$ sudo grep xkrum003 /etc/sub{gid,uid}
```

Jehož výsledkem je například:

```
/etc/subgid:xkrum003:165536:65536
/etc/subuid:xkrum003:165536:65536
```

Kde je první číslo je id prvního poduživatele a druhé je jejich možný počet. Tyto hodnoty se využijí při vytváření výchozího nastavení všech kontejnerů. Tím je dosaženo toho, že každý vytvořený kontejner má uživatele *root* mapován na unikátního poduživatele. Tento konfigurační soubor je nutné vytvořit v domovském adresáři uživatele (**~/.config/lxc/default.conf**), pod kterým bude kontejner spuštěn. Minimálně musí obsahovat následující údaje: [32] [33]

```
lxc.include = /etc/lxc/default.conf
lxc.id_map = u 0 165536 65536
lxc.id_map = g 0 165536 65536
```

Parametr	**Popis**
lxc.include	Cesta k výchozí konfiguraci pro všechny uživatele. Nachází se typicky v */etc/lxc/default.conf*
lxc.id_map	Seznam oddělený mezerami obsahující typu dat (u – uživatel, g – skupina), id v kontejneru, první povolené id na hostovaném systému a počet možných uživatelů, respektive skupin.

Tabulka 8 - Vybrané parametry konfigurace LXC [11]

V této chvíli je možné stáhnout a spustit kontejner odpovídající architektuře hostovaného systému. Kontejner pro architekturu amd64 nelze nativně spustit na i386 z technických důvodů. Za pomoci technologií jako *qemu* lze ale s omezeními spustit kontejnery i pro jiné architektury. [34]

V rámci této práce je využit kontejner obsahující stejnou distribuci jako hostující OS. Díky tomu odpadá možný problém s nekompatibilitou jádra systému:

```
$ lxc-create -t download --name dp-lxc

Distribution: ubuntu
Release: xenial
Architecture: i386
```

Obsah kontejneru je uložen v adresáři **~/.local/share/lxc**. Zde se nachází celý extrahovaný souborový systém kontejneru a soubor záznamů. Kontejner je spuštěn pomocí příkazu *lxc-create* s parametrem d:

```
$ lxc-start –name dp-lxc -d
```

Tento parametr vynutí spuštění na pozadí, v opačném případě by došlo k přepnutí na přihlašovací obrazovku v kontejnerovaném systému. Bylo by pak nutné vypnout celý kontejner přes jiný terminál, protože z bezpečnostních důvodů neobsahují kontejnerované systémy jiné uživatele než *root*. Samotný uživatel *root* nemá nastavené heslo. Je tak možné se do kontejneru přihlásit pouze příkazy *lxc-attach* a *lxc-console*.

LXC během své instalace vytváří takzvaný síťový most, který umožňuje tok dat, mezi několika síťovými rozhraními, díky tomu je možné spojení kontejnerů s vnější sítí. Bohužel se při NATování počítá pouze s protokolem IPv4. V případě, že je server dostupný pouze na IPv6, je situace složitější. Jednak je cílem mít kontejnery dostupné nepřímo, přes hostovaný operační systém a také LXC není na IPv6 úplně připraveno. Výsledkem je nedostupnost sítě Internet z kontejneru. Jelikož se v případě IPv6 s NATem nepočítá, je nutné využít aplikací jako je *radvd*. Ty přiřazují kontejnerům a dalším aplikacím adresy, které jsou dostupné na rozhraní vnější sítě. [35]

Vzhledem k obtížím s IPv6 bylo v případě této práce zvoleno řešení s povoleným IPv4, které je pro kontejnery NATované přes rozhraní na hostovaném operačním systému.

Nejprve je nutné povolit přeposílání paketů, forwarding, na hostovaném stroji. Toho bylo docíleno úpravou souboru **/etc/ufw/sysctl.conf**. Změněné parametry jsou následující:

```
net/ipv4/ip_forward=1
net/ipv6/conf/default/forwarding=1
net/ipv6/conf/all/forwarding=1
```

Parametr	**Popis**
net/ipv4/ip_forward	Povolení směrování paketů protokolu IPv4
net/ipv6/conf/default/forwarding	Výchozí nastavení směrování paketů pro nová IPv6 rozhraní
net/ipv6/conf/all/forwarding	Výchozí nastavení směrování paketů pro všechna IPv6 rozhraní

Tabulka 9 - Vybrané parametry nastavení sítě na úrovni jádra OS [36]

Tento soubor přepisuje hodnoty uvedené v souboru **/etc/sysctl.conf**, který je dostupný i v jiných distribucích.

Tyto změny umožňují přeposílat provoz na další rozhraní přes síťové mosty na úrovni operačního systému. Následně je nutné nastavit a povolit konektivitu ze sítě kontejnerů do Internetu skrze rozhraní hostujícího systému. Změna hodnoty DEFAULT_FORWARD_POLICY na ACCEPT v souboru **/etc/default/ufw** povolí pakety, které nejsou určeny pro rozhraní, na které přijdou.

Jelikož jsou servery připojeny přes IPv6, je pro povolení této konektivity povolit provoz i na úrovni firewallu.

```
IPV6=yes
```

Výchozí chování pro přesměrování, příchozí a odchozí provoz, je možné nastavit na ACCEPT a DROP, což nastavuje povolení, respektive zahození paketů. Z bezpečnostního hlediska je podstatné zahazování příchozího provozu. Povolení odchozího provozu a přesměrování vychází z využití vnější sítě při instalaci a použití kontejnerů.

```
DEFAULT_INPUT_POLICY="DROP"
```

```
DEFAULT_OUTPUT_POLICY="ACCEPT"
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Vzhledem k tomu, že UFW v instalované verzi korektně neaktualizuje pravidla pro směrování paketů a další *iptables* pravidla, je nutné nastavit parametr MANAGE_BUILTINS na *yes*. V případě používání pouze UFW to není problém, pokud ale pravidla pro *iptables* upravuje i jiná aplikace, je nutné případné konflikty a vícenásobně zmíněná pravidla řešit manuálně. [37]

```
MANAGE_BUILTINS=yes
```

Parametr	**Popis**
IPV6	Povolení protokolu IPv6
DEFAULT_INPUT_POLICY	Výchozí chování pro pakety z WAN
DEFAULT_OUTPUT_POLICY	Výchozí chování pro pakety do WAN
DEFAULT_FORWARD_POLICY	Výchozí chování pro pakety určené pro jiná rozhraní
DEFAULT_APPLICATION_POLICY	Výchozí nastavení pro aplikační profily
MANAGE_BUILTINS	Nastavení, zda má UFW spravovat cizí řetězy pravidel (například pravidla *iptables*)
IPT_SYSCTL	Nastavení cesty k dodatečné konfiguraci *iptables*

Tabulka 10 - Vybrané parametry nastavení UFW [38]

Následně je nutné nastavit firewall tak, aby přeposílal pakety ze sítě LXC kontejnerů přes rozhraní *eth0*, které slouží pro připojení do sítě Internet. Bohužel toto pravidlo nelze realizovat přes *ufw* a tak je nutné změny zapsat přímo jako pravidla *iptables* do souboru **/etc/ufw/before.rules**. Zde uvedená pravidla jsou použita před samotným spuštěním *ufw* a díky tomu jsou včas aplikována. Na začátek souboru je tedy nutné připsat následující příkazy:

```
*nat
:POSTROUTING ACCEPT [0:0]

-A POSTROUTING -s 10.0.3.0/24 -o eth0 -j MASQUERADE
```

```
COMMIT
```

Adresa 10.0.3.0/24 je adresou IPv4 s maskou, kterou využívá LXC pro své kontejnery. Alternativně lze využít i jméno síťového mostu, což by bylo v tomto případě *lxcbr0*. Zjistit adresu lze příkazem:

```
$ ip a
```

Jehož výpis je například následující:

```
3: lxcbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc noqueue state UP group default qlen 1000
    link/ether 00:16:3e:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 10.0.3.1/24 scope global lxcbr0
       valid_lft forever preferred_lft forever
    inet6 fe80::216:3eff:fe00:0/64 scope link
       valid_lft forever preferred_lft forever
```

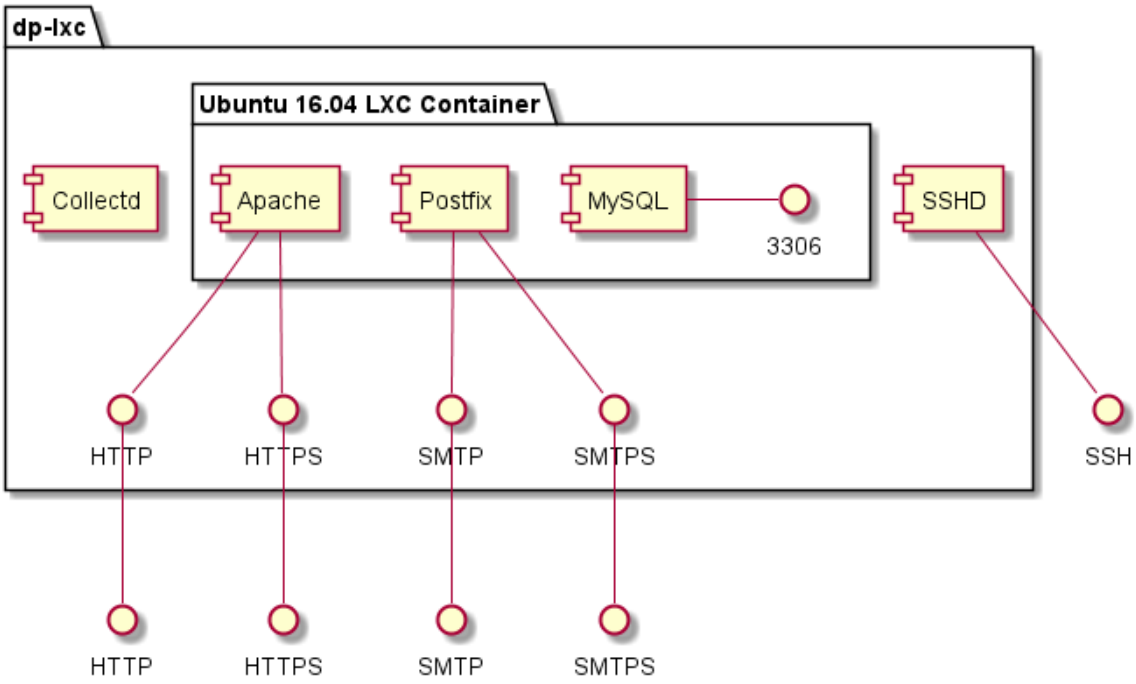
Kde je za klíčovým slovem *inet* IPv4 adresa rozhraní a slouží jako síťová brána pro kontejnery.

Po dokončení nastavení firewallu je nutné jej vypnout a zapnout:

```
$ sudo ufw disable && sudo ufw enable
```

Pro automatické spouštění LXC kontejnerů je nutné vytvořit odkaz v adresáři **/etc/lxc/auto/**, který vede na konfigurační soubor kontejneru. [38]

```
$ ln -s /var/lib/lxc/dp-lxc/config /etc/lxc/auto/dp-lxc.conf
```



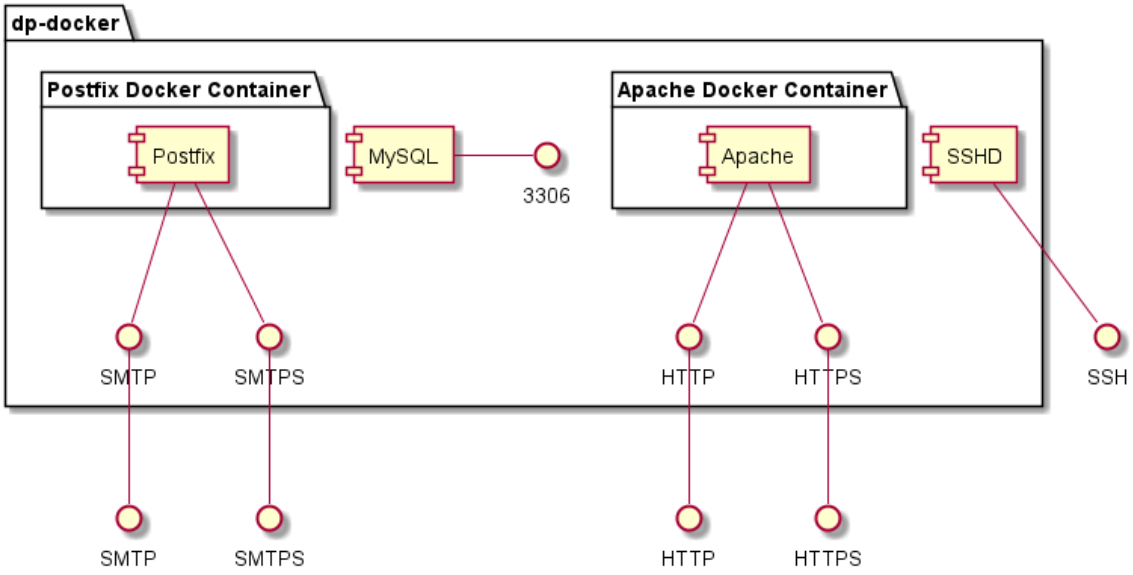
Obrázek 7 - Schéma instalovaných služeb s LXC kontejnery

4.2.2 Docker

Na rozdíl od LXC je vhodné vytvořit pro každou službu vlastní kontejner. Cílem je tedy spuštění kontejnerů pro mailserver a webserver s tím, že budou využívat databázový server spuštěný na hostovaném operačním systému.

V Docker repositářích existuje velké množství předpřipravených obrazů, které lze využít přímo nebo rozšířit pomocí *Dockerfile* souborů. V rámci zachování co největší míry podobnosti s jinak instalovanými službami, jsou upřednostňovány obrazy využívající základní Docker obrazy Ubuntu:16.04.

Jednotlivé konfigurace Docker kontejnerů jsou uvedeny v rámci konfigurace příslušných služeb.



Obrázek 8 - Schéma instalovaných služeb s Docker kontejnery

Stejně jako v případě instalace LXC kontejnerů je nutné povolit směrování paketů mezi rozhraními jak na úrovni jádra, tak na úrovni *ufw*. Je tedy nutné změnit parametry v souboru **/etc/ufw/sysctl.conf** na následující hodnoty:

```
net/ipv4/ip_forward=1
net/ipv6/conf/default/forwarding=1
net/ipv6/conf/all/forwarding=1
```

Popis parametrů viz Tabulka 9 - Vybrané parametry nastavení sítě na úrovni jádra OS [36].

A následně povolit směrování dat ve firewallu v souboru **/etc/default/ufw**:

```
IPV6=yes
DEFAULT_INPUT_POLICY="DROP"
DEFAULT_OUTPUT_POLICY="ACCEPT"
DEFAULT_FORWARD_POLICY="ACCEPT"
MANAGE_BUILTINS=yes
```

Popis parametrů viz Tabulka 10 - Vybrané parametry nastavení UFW [38].

4.3 Instalace a konfigurace služeb

Instalace služeb probíhá shodně v případě nekontejnerové varianty a instalace v LXC kontejneru. V případě Docker kontejnerů je instalační skript součástí konfiguračního souboru kontejneru a obsahuje podobné příkazy, které se mohou lišit v případě využití kontejneru založeném na jiné distribuci. Z tohoto důvodu je instalace Docker kontejnerů zmíněna zvlášť.

Konfigurace jednotlivých služeb je z velké části shodná a přenositelná mezi instalacemi v kontejneru i přímo na hostovaném systému. Konfigurační direktivy jsou zde tedy uvedeny jen jednou a případné odchylky jsou explicitně zmíněny.

4.3.1 Databázový server

V použité distribuci Ubuntu je MySQL server nainstalován přes balíčkovací systém *apt*. V rámci instalace je nutné zadat heslo pro uživatele *root*. Alternativně je možné dokončit nastavení databázového serveru před jeho spuštěním příkazem *mysql_secure_installation*.

Veškeré konfigurační soubory se nacházejí v adresáři **/etc/mysql/**. Základní nastavení MySQL serveru vypadá následovně:

```
[mysqld]
user    = mysql
socket  = /var/run/mysqld/mysqld.sock
port    = 3306
datadir = /var/lib/mysql
```

Ve výchozím nastavení je MySQL server dostupný na portu *3306*. Jelikož není žádoucí mít server dostupný ze sítě Internet na veřejných IP adresách, je parametr *bind-address* nastaven na hodnotu *127.0.0.1*. Tato IP adresa též odpovídá doménovému jménu *localhost*. Alternativně je také dostupný přes socket **/var/run/mysqld/mysqld.sock**.

```
bind-address = 127.0.0.1
```

Parametr	**Popis**
user	Uživatel, pod kterým je server spuštěn
port	Port, na kterém je server dostupný na adrese, respektive adresách uvedených v *bind-address*
datadir	Cesta k souborům samotných databází
bind-address	Adresa, či skupina adres, na kterých je server dostupný. Je možné zde nastavit nejen adresu jednoho rozhraní, ale také případně všechny rozhraní (*), s protokolem IPv4 (0.0.0.0), nebo IPv6 (::)

Tabulka 11 - Vybrané konfigurační parametry serveru MySQL [21]

4.3.1.1 Docker

V případě využití MySQL v Docker kontejneru je nutné nastavit parametr *bind-address* dle nastavení sítě, která je společná pro kontejnery a hostující systém.

Ve výchozím nastavení není možnost přihlásit se na uživatele root z jiné než lokální adresy. V případě MySQL serveru vně kontejneru je tak nutné z hostujícího operačního systému přidat uživatele, který umožní přístup z IP adresy kontejneru. V tomto případě má kontejner IP adresu 172.17.0.2 a přidání uživatele root se všemi oprávněními pro IP kontejneru lze následujícími příkazy. Poslední příkaz aplikuje změny:

```
CREATE USER 'root'@'172.17.0.2' IDENTIFIED BY
'password';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'172.17.0.2' WITH
GRANT OPTION;

FLUSH_PRIVILEGES;
```

Heslo *password* je uvedeno pouze pro názornost.

4.3.2 Webový server

Široce používaný webový server Apache je součástí dostupných balíčku přes aplikaci *apt*. Samotný Apache je schopný poskytovat přístup ke statickým webovým stránkám, což je pro účely testování výkonu nedostačující. Z tohoto důvodu je také instalován interpret jazyka PHP, ve kterém budou testované webové stránky napsané.

Konfigurace Apache se nachází v adresáři a podadresářích **/etc/apache2/** a hlavní konfigurační soubor se nachází v **/etc/apache2/apache2.conf**.

S ohledem na bezpečnost je vhodné omezit informace, které o sobě webový server poskytuje, jelikož ve výchozím nastavení informuje klienta o své verzi a operačním systému. Toho mohou potenciálně využít útočníci, pokud by byla nalezena chyba v aktuálně instalované verzi. Toto chování se řídí parametry *ServerTokens* a *ServerSignature*. Hodnota *Prod* v prvním parametru zajišťuje, že se bude odesílat pouze název Apache. Z této hodnoty mohou poté vycházet statistiky využití webserverů na internetu a zároveň se minimalizuje množství informací poskytovaných klientům, respektive útočníkům. Parametr *ServerSignature* nastavuje vkládání podpisu serveru, obsahující jeho verzi, ve stránkách generovaných serverem jako jsou chybové stránky, FTP výpis souborů apod. [39] Nové hodnoty v souboru **/etc/apache2/conf-available/security** tak jsou následující:

```
ServerTokens Prod
ServerSignature Off
```

Velká část bezpečnosti webserverů souvisí s webovými aplikacemi, ke kterým poskytuje přístup. Na úrovní serveru je v současné době doporučené povolit a vynutit protokol *HTTPS*. Komunikace v rámci *HTTP* je pak využita pouze pro přesměrování klienta na šifrovanou verzi protokolu.

Veškeré nastavení spojené s šifrováním HTTPS, kromě nastavení cest k certifikátům, je možné provést globálně. Bezpečné kombinace šifrovacích algoritmů poskytuje například Mozilla ve svém Mozilla SSL Configuration Generator. Tento generátor nabízí 3 skupiny povolených šifer s tím, že čím je kombinace bezpečnější, tím menší bude jejich podpora u klientů. Volba *Intermediate* poskytuje dostatečnou úroveň zabezpečení a zároveň nabízí dostatečně širokou podporu i pro starší prohlížeče jako je Internet Explorer 8 a další. [40] Výsledná konfigurace *HTTPS* se nachází v souboru **/etc/apache2/mods-available/ssl.conf**.

Konfigurace SSL/TLS parametrů pro Apache modul je úzce svázána s používanou kryptografickou knihovnou. V operačním systému Linux je nejpoužívanější knihovna OpenSSL, která poskytuje podporu pro mnoho šifrovacích algoritmů a protokolů. Mezi protokoly, které jsou podporovány, ale nedoporučovány z důvodu bezpečnosti je aktuálně také SSLv3, který je vhodný zakázat. Jelikož se ale předpokládá dokončení specifikace protokolu TLSv1.3, je vhodnější jmenovat špatné protokoly než ty bezpečné. Odpadne tím nutnost dopisovat nově implementované protokoly v budoucnu. Hodnota *all* zastupuje všechny dostupné protokoly a modifikátor „-„ (minus) umožňuje vyjmout ze seznamu jmenovanou hodnotu. [41]

```
SSLProtocol all -SSLv3
```

Seznam šifer vychází z doporučení společnosti Mozilla. Pokud není vyhovující seznam šifer k dispozici, lze v tomto parametru využít zápisy pro skupiny šifrovacích algoritmů. Vykřičníkem označené algoritmy a skupiny jsou zakázány. S ohledem na parametr *SSLHonorCipherOrder* je vhodné řadit algoritmy od nejbezpečnějšího. [41]

```
SSLCipherSuite ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-
CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-
RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-
SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-
ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-
AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-
CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-
SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-
SHA:!DSS
```

Jelikož jsou šifrovací algoritmy seřazeny podle úrovně bezpečnosti, je doporučované vynutit tuto preferenci oproti preferenci udané klientem. V opačném případě by klient mohl preferovat slabší šifrovací algoritmy, na které je jednodušší provést útok typu MitM, nebo odposlech. [41]

```
SSLHonorCipherOrder on
```

SSL komprese umožňuje zmenšit hlavičky paketů. Bohužel tato optimalizace umožňuje v některých případech dešifrovat odesílaná data, a proto se doporučuje jí vypínat. [42]

```
SSLCompression  off
```

Parametr	**Popis**
SSLProtocol	Seznam povolených SSL/TLS protokolů
SSLCipherSuite	Seznam povolených šifrovacích algoritmů
SSLHonorCipherOrder	Nastavení vynucení pořadí šifrovacích algoritmů serverem
SSLCompression	Povolení komprese SSL komunikace

Tabulka 12 - Vybrané konfigurační parametry Apache2 pro mod_ssl [41]

Pro zprovoznění *HTTPS* je nutné povolit moduly *mod_ssl* a *mod_socache_shmcb* vytvořením odkazu na jejich konfigurační v adresáři **/etc/apache2/mods-enabled/**.

```
$ sudo ln -s ../mods-available/ssl.load
$ sudo ln -s ../mods-available/ssl.conf
$ sudo ln -s ../socache_shmcb.load
$ sudo systemctl reload apache2
```

4.3.2.1 Docker

Při instalaci Apache Docker kontejneru byly využity již připravené konfigurační soubory. Z důvodu následného testování výkonu využívá kontejner z repositářů pouze obraz Ubuntu 16.04 LTS. Tento obraz je rozšířen o nainstalovaný Apache server se stejnými moduly, jako v případě bezkontejnerové implementace i instalace v LXC kontejneru.

Výsledný *Dockerfile* soubor obsahuje instrukce pro instalaci závislostí, import konfiguračních souborů a povolení modulů při spuštění. Jelikož je v Docker kontejneru spuštěn jen Apache server, provádí se spouštění přes příkaz *apachectl*. Aplikace pro správu systému jako je třebas *systemd* a jeho *service* nejsou potřebné. Poslední instrukce zpřístupní na kontejneru porty 80 a 443 pro HTTP a HTTPS:

```
FROM ubuntu:16.04

RUN export DEBIAN_FRONTEND=noninteractive && apt-get
update && apt-get upgrade -y
RUN export DEBIAN_FRONTEND=noninteractive && apt-get
install apache2 libapache2-mod-php php-mcrypt php-mysql
-y

COPY apache2.conf /etc/apache2/
COPY ports.conf /etc/apache2/
COPY security.conf /etc/apache2/conf-available/
COPY ssl.conf /etc/apache2/mods-available/
COPY socache_shmcb.load /etc/apache2/mods-available/
COPY default-ssl.conf /etc/apache2/sites-available/

RUN ln -s /etc/apache2/mods-available/socache_shmcb.load
/etc/apache2/mods-enabled/socache_shmcb.load
RUN ln -s /etc/apache2/mods-available/ssl.conf
/etc/apache2/mods-enabled/ssl.conf
RUN ln -s /etc/apache2/mods-available/ssl.load
/etc/apache2/mods-enabled/ssl.load

RUN ln -s /etc/apache2/sites-available/default-ssl.conf
/etc/apache2/sites-enabled/default-ssl.conf
```

```
CMD ["apachectl", "-D", "FOREGROUND"]

EXPOSE 80 443
```

Tento kontejner je následně sestaven jako *xkrum003/apache* a spuštěn pod jménem *apache* následujícími příkazy:

```
$ docker build -t xkrum003/apache .
$ docker run -d -p 80:80 -p 443:443 --name apache
xkrum003/apache
```

4.3.3 Mailový server

V případě této práce je instalována pouze MTA část mailového serveru, kterou zajišťuje často využívaný software Postfix.

Server Postfix umožňuje ukládat emailové zprávy ve dvou formátech:

- *Mailbox* – jedná se o starší formát, kdy se veškeré uložené emailové zprávy nachází v jednom souboru pro každý emailový účet. Ze své podstaty je nevhodný pro větší množství emailů z důvodu rychlosti přístupu k jednotlivým zprávám a jejich horší správě. Tento formát je použit, pokud cesta k úložišti emailů nekončí lomítkem. [18] [43]
- *Maildir* – v případě tohoto formátu jsou emailové zprávy uloženy v adresáři, respektive podadresářích podobně, jako jsou vidět přes protokol IMAP. Z důvodu lepší přehlednosti, správě a jednoduššího přístupu se jedná o upřednostňovaný způsob ukládání zpráv. [18] [43]

Veškeré emailové účty jsou virtuální a informace o nich jsou vedeny v databázi serveru MySQL. Z tohoto důvodu je nutné vytvořit příslušnou databázi s účtem, který bude Postfix využívat pro přístup k datům o emailových účtech:

```
mysql> CREATE DATABASE `postfix`;
mysql> CREATE USER 'postfix'@'localhost' IDENTIFIED BY
'password'
mysql> GRANT ALL PRIVILEGES ON postfix.* TO
'postfix'@'localhost';
```

V případě Docker kontejneru je místo *localhost* uvedena IP adresa kontejneru, na kterém běží aplikace Postfix. Heslo password je uvedeno pouze pro názornost.

Instalaci Postfix serveru lze provést přes příkaz *apt-get*. Konfigurační soubory pro Postfix jsou uloženy v adresáři **/etc/postfix/**. V hlavním konfiguračním souboru **main.cf** je nutné nastavit parametr my*hostname*, jedná se o doménovou adresu, na které je emailový server dostupný. Hodnota tohoto parametru dále využívána v dalších konfiguračních direktivách ve formě zápisu *$myhostname*. Dále předpokládáme dostupnost emailových služeb na všech rozhraních a protokolech IP s tím, že přijímat emaily budeme pro všechny lokální domény a doménu emailového serveru.

```
inet_interfaces = all
inet_protocols = all
mydestination = localhost
relay_domains = $mydestination
```

Hodnota *localhost* v parametru *mydestination* je plně postačující, protože seznam domén, které mailserver spravuje bude uložena v databázi. Pokud by tomu tak nebylo, bylo by nutné seznam domén zapsat do tohoto parametru.

Odesílat emaily ale je v tomto případě povoleno pouze z vnitřních adres, respektive adres kontejnerů. Jedná se o bezpečnější nastavení v případě, kdy není potřeba odesílat emaily přes tento server například z emailového klienta z vnější sítě. Toho lze docílit buď vypsáním všech možných lokálních adres v parametru *mynetworks*, nebo nastavením šablony sítí. V tomto případě odpovídá šablona *host*.

```
#mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104
[::1]/128
mynetworks_style = host
```

Parametr	**Popis**
myhostname	FQDN, na které je Postfix dostupný
mydomain	Doména, na které je Postfix dostupný

inet_interfaces	Síťová rozhraní, na kterých Postfix naslouchá
inet_protocols	Internetový protokol, na kterém Postfix naslouchá
mydestinations	Adresy, pro které bude Postfix cílovým serverem
relay_domains	Adresy, pro které bude Postfix přeposílat emaily na další servery
mynetworks	Seznam IP rozsahů, které jsou považovány za bezpečné a na které se vztahují menší bezpečnostní omezení
mynetworks_style	Název šablony IP rozsahů, která nahrazuje zápis parametru mynetworks.

Tabulka 13 - Vybrané základní konfigurační parametry Postfix [43]

K virtuálním schránkám je možné nastavit omezení velikosti. To lze nastavit parametrem *virtual_mailbox_limit*, jehož hodnota je velikost v bajtech. Následující nastavení omezuje velikost schránky na 50 MB:

```
virtual_mailbox_limit = 52428800
```

Účty, které nejsou vázané na reálně uživatelské účty v systému mohou být uloženy kdekoli, v případě této práce je použita složka **/var/mail/vhosts/**. Ve výchozím nastavení se Postfix snaží nastavit vlastníka souborů na reálného uživatele. Jelikož je obvykle pro správu emailů uživateli využit jiný program, například webové rozhraní nebo MUA Dovecot, je vhodné nastavit jako vlastníka jeden jediný účet. V tomto případě je využito *uid* a *gid* uživatele postfix:

```
virtual_uid_maps = static:113
virtual_gid_maps = static:121
virtual_mailbox_base = /var/mail/vhosts
virtual_transport = virtual
```

Jelikož je jako úložiště informací o emailových schránkách, doménách a přesměrování, je nutné nastavit příslušné parametry pro virtuální emailové schránky skrze Postfix plugin *mysql*:

```
virtual_mailbox_domains = mysql:/etc/postfix/mysql-
virtual-mailbox-domains.cf
virtual_mailbox_maps = mysql:/etc/postfix/mysql-virtual-
mailbox-maps.cf
```

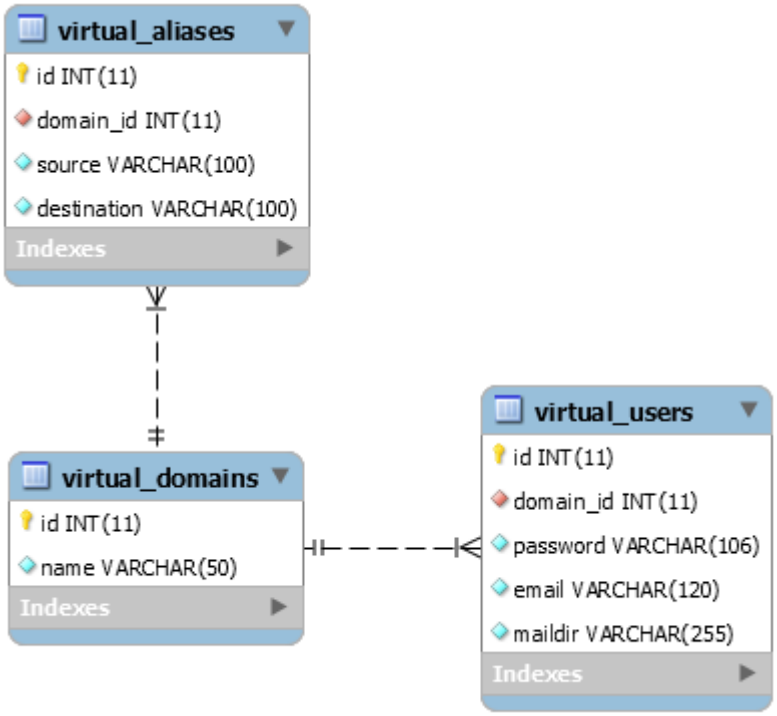
```
virtual_alias_maps = mysql:/etc/postfix/mysql-virtual-
alias-maps.cf
```

Výše zmíněné soubory obsahují nastavení dotazů do databáze pro MySQL plugin tak, aby poskytoval relevantní výstupy z databáze. I když je databáze přístupná na adrese *localhost*, je v konfiguračních souborech nastavena IP 127.0.0.1. Je to z důvodu chování Postfix MySQL pluginu, kdy se na adresy *localhost* připojuje přes socket. S ohledem na jednotnou konfiguraci a možná různá umístění *socket* souboru je upřednostňováno připojení přes port. [43]

Parametr	**Popis**
virtual_mailbox_limit	Omezení emailové schránky v bajtech
virtual_uid_maps	ID vlastníka souborů emailů
virtual_gid_maps	ID skupiny vlastníka souborů emailů
virtual_mailbox_base	Cesta k souborům emailů v souborovém systému
virtual_transport	Nastavení subsystému, který zpracuje příchozí zprávy
virtual_mailbox_domains	Seznam, nebo zdroj seznamu, domén spravovaných emailovým serverem
virtual_mailbox_maps	Seznam, nebo zdroj seznamu, cest k souborům emailů pro jednotlivé emaily
virtual_alias_maps	Seznam, nebo zdroj seznamu, aliasů jednotlivých emailových účtů

Tabulka 14 - Vybrané konfigurační parametry virtuálních emailových účtů Postfix [43]

Veškerá konfigurace zmíněná výše vychází ze struktury databáze a tato struktura není striktně definována. Pro potřeby této práce je použita jednoduchá databáze o 3 tabulkách pro vedení informací o spravovaných doménách, uživatelích a aliasech. Z této struktury poté vychází konfigurace pro Postfix. Diagram struktury této databáze je na následujícím obrázku.



Obrázek 9 - Struktura MySQL databáze pro Postfix

Tabulka *virtual_aliases* slouží pro ukládání informací o emailových aliasech. Tyto aliasy jsou vázány na virtuální domény, které emailový server obsluhuje. Ty jsou uloženy v tabulce *virtual_domains*. K doménám jsou také přiřazeny uživatelské účty, respektive emaily, ty jsou uloženy v tabulce *virtual_users*. Celá struktura této databáze je k dispozici ve formě SQL dotazu v příloze.

V případě *virtual_mailbox_domains* se jako výstup očekává název domény, pokud je nalezena. [43] Soubor **mysql-virtual-mailbox-domains.cf** tak obsahuje následující:

```
user = postfix
password = password
hosts = 127.0.0.1
dbname = postfix
table = virtual_domains
select_field = name
where_field = name
```

V případě *virtual_mailbox_maps* se očekává název emailu, nebo jiný textový řetezec. Ten je poté využit při vytváření názvu souboru obsahujícího doručené emaily v adresáři, který je uveden v parametru *virtual_mailbox_base*. V případě, že výsledná cesta

končí lomítkem, je použit formát Maildir, v opačném případě jsou veškeré emaily ukládané do jednoho souboru formátu Mailbox. [43] Obsahem souboru **mysql-virtual-mailbox-maps.cf** je následující konfigurace:

```
user = postfix
password = password
hosts = 127.0.0.1
dbname = postfix
table = virtual_users
select_field = maildir
where_field = email
```

Parametr *virtual_alias_maps* očekává aliasovaný email. [43] Soubor **mysql-virtual-alias-maps.cf** obsahuje následující:

```
user = postfix
password = password
hosts = 127.0.0.1
dbname = postfix
table = virtual_aliases
select_field = destination
where_field = source
```

Správnost konfigurace přístupu k datům v databázi je možné otestovat příkazem *postmap*, kdy se za parametr *-q* zadá hledaná doména, respektive email. V případě úspěchu je vrácen výsledný textový řetězec z databáze:

```
$ postmap -q xkrum003@dp-bare.duke-hq.net
mysql:/etc/postfix/mysql-virtual-mailbox-maps.cf

dp-bare.duke-hq.net/xkrum003/
```

V tomto případě je vrácena relativní cesta k emailovým souborům daného účtu k cestě uvedené v parametru *virtual_mailbox_base*.

4.3.3.1 Docker

Vzhledem k tomu, že Postfix neumožňuje spuštění na popředí a díky tomu také sledování stavu kontejneru, je nutné využít supervizory a další podobné aplikace. [44] Z tohoto důvodu je výhodnější využít již existující a fungující šablonu kontejneru a tu upravit. Šablona *postfix:latest* byla zvolena z důvodu kompatibility verzí i distribuce

Linux. Do této šablony jsou vloženy upravené konfigurační soubory, které jsou využity při instalaci bez kontejneru a v kontejneru LXC. **Dockerfile** tohoto kontejneru vypadá následovně:

```
From postfix:latest

COPY mysql-virtual-mailbox-domains.cf /etc/postfix/
COPY main.cf /etc/postfix/
COPY mysql-virtual-mailbox-maps.cf /etc/postfix/
COPY mysql-virtual-alias-maps.cf /etc/postfix/
```

Kontejner je poté sestaven a spuštěn s tím, že úložiště emailů je z důvodu persistence propojeno s adresářem na hostujícím operačním systému. Z důvodu možnosti kontroly chyb je adresář pro logy propojen do adresáře logů hostujícího OS:

```
$ docker build -t xkrum003/postfix .
$ docker run -d -p 25:25 -p 465:465 -p 587:587 -
v /var/mail/vhosts:/var/mail/vhosts -
v /var/log/postfix:/var/log/postfix postfix
```

Alternativně lze nastavení portů a připojení adresářů nastavit v souboru *docker-compose.yml*.

4.3.4 Firewall

Pro zpřístupnění služeb ze sítě Internet je nutné povolit porty na rozhraní *eth0* hostujícího operačního systému, respektive operačního systému bez kontejnerů. Vzhledem k poskytovaným službám je nutné povolit, případně směrovat, porty 80 (HTTP), 443 (HTTPS), 25 (SMTP),465 a 587 (SMTPS). Dále je nutné povolit port 22 (SSH) pro umožnění vzdálené správy. Ve všech případech je povolen pouze příchozí provoz protokolem TCP.

Otevření portů lze provést přímo přes *ufw*. V případě SSH je možné využít povolení aplikace:

```
$ sudo ufw allow 'Apache Full'
$ sudo ufw allow 'OpenSSH'
```

Komunikaci na portech lze povolit uvedením jeho čísla nebo názvu služby, která je uvedená v souboru **/etc/services**. Porty pro HTTP/S a SMTP/S lze porty povolit následujícím způsobem:

```
$ sudo ufw allow http
$ sudo ufw allow https
$ sudo ufw allow smtp
$ sudo ufw allow smtps
```

Porty jsou díky těmto příkazům připraveny k otevření na protokolech IPv4 i IPv6. Pro aktivaci nastavení je nutné jej načíst:

```
$ sudo ufw reload
```

4.3.4.1 LXC

V případě využití LXC kontejnerů je nutné zajistit směrování portů z vnějšího rozhraní *eth0* na port LXC kontejneru. Pro toto je nutné upravit soubor **/etc/ufw/before.rules** a do skupiny *nat* přidat *iptables* příkazy:

```
-A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT --to
10.0.3.183:80
-A PREROUTING -i eth0 -p tcp --dport 443 -j DNAT --to
10.0.3.183:443
-A PREROUTING -i eth0 -p tcp --dport 25 -j DNAT --to
10.0.3.183:25
-A PREROUTING -i eth0 -p tcp --dport 465 -j DNAT --to
10.0.3.183:587
-A PREROUTING -i eth0 -p tcp --dport 587 -j DNAT --to
10.0.3.183:587
```

Kde IP adresa 10.0.3.183 je adresa kontejneru, kterou lze zjistit příkazem *lxc-info*:

```
$ lxc-info --name thesis
Name:           thesis
State:          RUNNING
PID:            1628
IP:             10.0.3.183
Memory use:     379.23 MiB
KMem use:       0 bytes
Link:           vethPTOHM8
```

```
TX bytes:      3.11 MiB
RX bytes:      3.86 MiB
Total bytes:   6.97 MiB
```

4.4 Sledování a vyhodnocení zátěže serveru

Měření výkonu bude řešeno aplikací Collectd. Ta nabízí velké množství pluginů pro získávání dat nejen o využití systémových prostředků, jako je procesorový čas, paměť atp., ale také podporuje sledování stavu mnoha aplikací. Výsledkem jsou data, která lze analyzovat jako časové řady. [31]

Pro porovnání náročnosti kontejnerových technologií byly zvoleny jako důležité parametry využití místa na souborovém systému, paměti a procesorového času. Tyto časové řady poté budou porovnány s cílem zjistit zda, a jak moc, kontejnerové technologie negativně ovlivňují výkon.

Jednotlivé instalace jsou testovány v situaci bez zátěže a poté se zátěží, jenž bude probíhat skrze testovací webovou aplikaci. Tato aplikace využívá všechny služby poskytované na serveru – databázi, webserver i mailserver.

4.4.1 Webová aplikace

Jako způsob generování zátěže byly zvoleny HTTP požadavky na webovou aplikaci, která je interpretovaná v jazyku PHP na instalovaném webserveru Apache s využitím databáze MySQL a mailserveru Postfix. Tento způsob byl zvolen s ohledem na podobnost s reálným provozem.

Testovací PHP aplikací byl zvolen projekt Wordpress verze 4.9.4 [45], s nímž má autor zkušenosti, jakožto plnohodnotný CMS. Wordpress poskytne dostatečné zatížení a umožní následné porovnání využití systémových prostředků při zátěži.

Na každé instanci aplikace Wordpress je pluginem FakerPress vygenerována stejně velká sada náhodných testovacích dat. Jedná se o uživatele a příspěvky tvořené náhodným, nesmyslným, textem. Plugin FakerPress je instalován ve verzi 0.4.11.

4.4.2 Generování zátěže

Generování zátěže je řešeno aplikací Apache jMeter 4 [46], který umožňuje paralelní zasílání HTTP požadavků na několik cílových adres. Díky tomu je možné rozložení plánované zátěže tak, aby testované implementace služeb byly rovnoměrně a stejně zatíženy.

Zátěž je generována HTTP GET požadavky na webovou adresu, kde je dostupný nainstalovaný CMS Wordpress s testovacími daty. Tyto požadavky jsou paralelně a rovnoměrně posílány na všechny instance webového serveru.

Samotné generování zátěže má tyto parametry pro každý server:

- 5 paralelních uživatelů přistupujících na web
- 600 zobrazených webových stránek každým uživatelem
- 1 sekunda mezi požadavky každého uživatele

4.4.3 Výsledky měření – stav bez zátěže

S ohledem na to, že byl stav bez zátěže dlouhodobý a nenarušovaný vnějšími vlivy, je pro analýzu zvolen interval jednoho dne. Charakteristiky určené pro porovnání jednotlivých implementací jsou medián, aritmetický průměr a rozptyl. Data jsou získána a analyzována pro den 19.3.2018. Před samotným měřením byly servery restartovány. Tato data jsou dostupná v příloze (Příloha 1 – DVD s konfiguračními soubory a kontejnery).

Číselná data o využité paměti je možné získat příkazem *rrdtool* pro každý instalovaný hostující OS. Tato data jsou následně analyzována v tabulkovém procesoru Excel.

Následující příkaz vrátí naměřené minutové hodnoty, respektive průměr z původních 10 sekundových v uvedeném období, z databáze dat o využití paměti RAM:

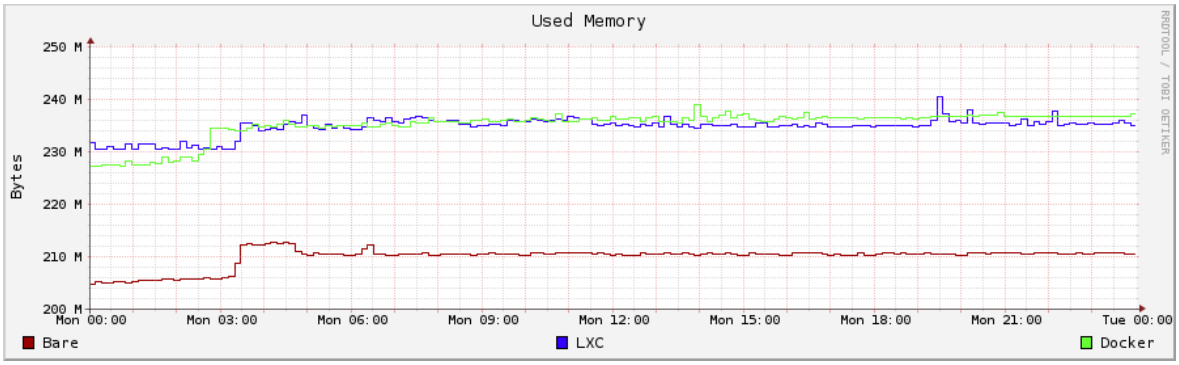
```
rrdtool fetch memory/memory-used.rrd AVERAGE -r 1m -
s '00:00 19.03.2018' -e '00:00 20.3.2018'
```

Naměřená data mají charakteristiky uvedené v následující tabulce.

	Bare	LXC	Docker
Průměr	209,95 MiB	234,75 MiB	235,17 MiB
Medián	210,48 MiB	235,13 MiB	236,11 MiB
Směrodatná odchylka	1,85 MiB	1,74 MiB	2,69 MiB

Tabulka 15 - Základní charakteristiky naměřených dat využití RAM bez zátěže

Důvodem pro vyšší směrodatnou odchylku je nárůst využití paměti u všech serverů zhruba ve 3 hodiny ráno. Tato změna je vidět na následujícím grafu a je způsobena údržbou záznamů a podobných pravidelných operací hostujícího operačního systému. Pro následné porovnání jednotlivých implementací jsou využity hodnoty mediánu.



Obrázek 10 - Grafické zobrazení naměřených dat využití RAM bez zátěže

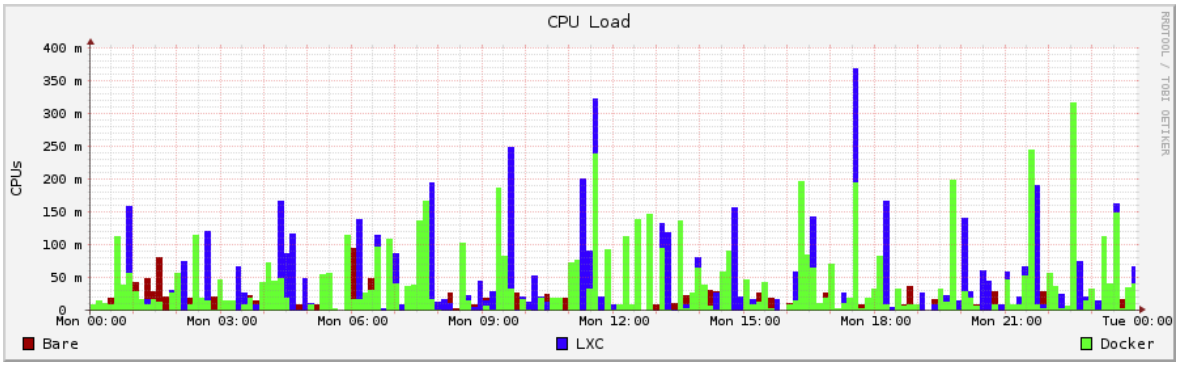
Průměrné minutové hodnoty z databáze dat o využití CPU jsou získány následujícím příkazem:

```
rrdtool fetch load/load.rrd AVERAGE -r 1m -s '00:00
19.03.2018' -e '00:00 20.3.2018'
```

	Bare	LXC	Docker
Průměr	0,015 CPU	0,036 CPU	0,037 CPU
Medián	0,009 CPU	0,010 CPU	0,015 CPU
Směrodatná odchylka	0,021 CPU	0,060 CPU	0,054 CPU

Tabulka 16 - Základní charakteristiky naměřených dat zatížení CPU bez zátěže

Díky k velkým výkyvům v malých hodnotách využití CPU je směrodatná odchylka relativně velká. Z toho důvodu jsou pro následné porovnání jednotlivých implementací využity hodnoty mediánu. Naměřená data jsou zobrazena na následujícím grafu, výkyvy jsou způsobeny operacemi jako je rotace záznamů, čištění paměti atp. a to jak v kontejnerech, tak v hostujícím systému.



Obrázek 11 - Grafické zobrazení naměřených dat zatížení CPU bez zátěže

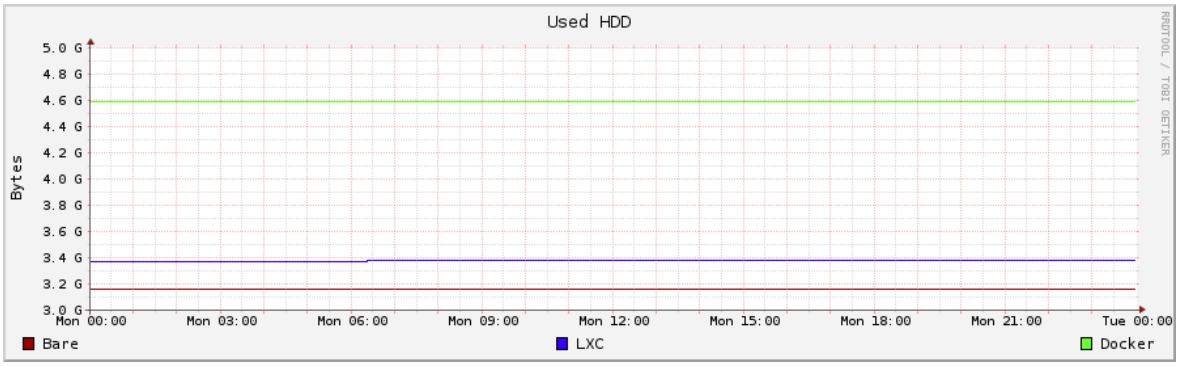
Průměrné minutové hodnoty z databáze dat o využití volného místa na disku jsou získány následujícím příkazem:

```
rrdtool fetch df-root/df_complex-used.rrd AVERAGE -r 1m
-s '00:00 19.03.2018' -e '00:00 20.3.2018'
```

	Bare	**LXC**	**Docker**
Průměr	3233,49 MiB	3458,41 MiB	4696,53 MiB
Medián	3233,48 MiB	3458,96 MiB	4696,43 MiB
Směrodatná odchylka	0,15 MiB	2,24 MiB	0,79 MiB

Tabulka 17 - Základní charakteristiky naměřených dat využití místa na HDD bez zátěže

Během tohoto období nebyla generována žádná zátěž, změny v zaplnění disků jsou tak způsobeny pouze generováním záznamů, jejich rotací záznamů, komprimací a případným mazáním. Jedná se ale o minimální změny vzhledem k celkovému zaplnění disku a tomu odpovídá i malá směrodatná odchylka. Pro porovnání jednotlivých kontejnerových technologií je využit medián z naměřených hodnot.



Obrázek 12 - Grafické zobrazení naměřených dat využití místa na HDD bez zátěže

4.4.4 Výsledky měření – stav se zátěží

Zátěž byla spuštěna zhruba v 13:10 a trvala 16 minut. Bylo při ní odesláno celkem 3000 požadavků na zobrazení stránky na každém instalovaném serveru. Všechny požadavky byly úspěšné a server vždy vrátil HTTP kód 200. Před testováním byly všechny virtuální servery restartovány. Získaná data jsou dostupní v příloze 1.

V případě zátěže jsou analyzována pouze data o využití paměti RAM a CPU. Velikost volného místa byla ovlivněna pouze zápisy záznamů operačního systému a webserveru. Analýza je následně provedena v tabulkovém procesoru Excel.

Pro analýzy byly použity interval naměřených desetisekundových hodnot v časech mezi 13:00 a 13:30. V případě využití paměti RAM byla data získána následujícím příkazem:

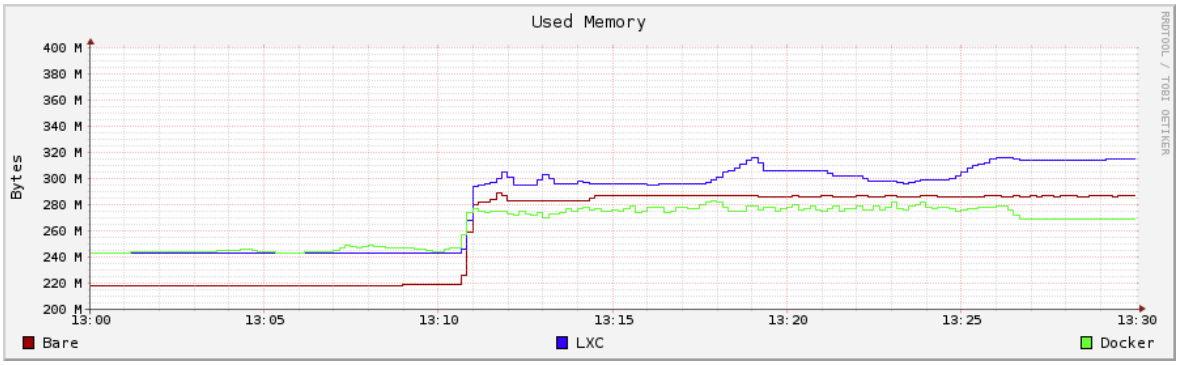
```
rrdtool fetch memory/memory-used.rrd AVERAGE -r 10s -
s '13:00 20.03.2018' -e '13:30 20.3.2018'
```

	Bare	**LXC**	**Docker**
Průměr	261,41 MiB	281,68 MiB	264,52 MiB
Medián	286,41 MiB	295,89 MiB	272,71 MiB
Směrodatná odchylka	32,41 MiB	29,63 MiB	14,88 MiB

Tabulka 18 - Základní charakteristiky naměřených dat využití RAM se zátěží

Vyšší směrodatná odchylka je dána rostoucím využitím paměti v čase ve všech případech. Pro porovnání jednotlivých implementací je využit medián dat, který odpovídá výsledné realitě více než průměr.

Nízké hodnoty pro virtuální server s Docker kontejnery je způsoben omezováním, které vynucuje Docker skrze kontrolní skupiny, *cgroupy*. Tento efekt je patrný také na naměřených datech využití CPU.



Obrázek 13 - Grafické zobrazení naměřených dat využití RAM se zátěží

Pro získání naměřených hodnot o využití CPU byl využit následující příkaz:

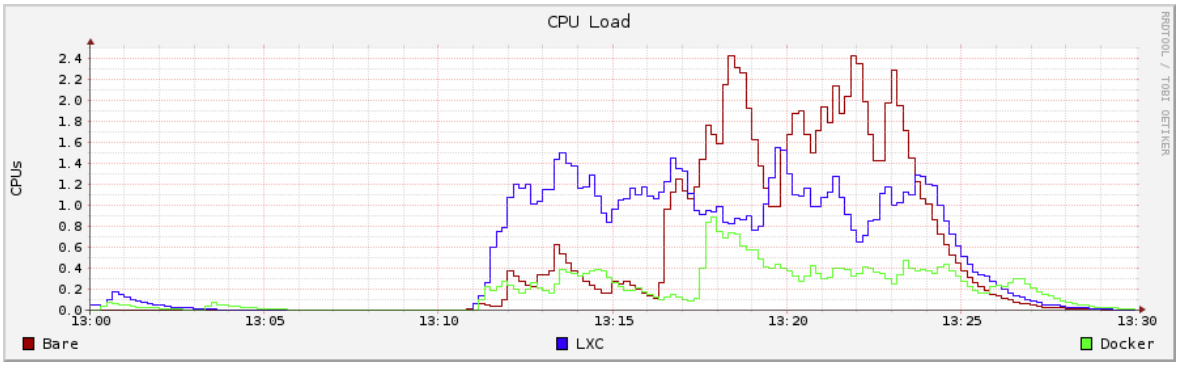
```
rrdtool fetch load/load.rrd AVERAGE -r 10s -s '13:00
20.03.2018' -e '13:30 20.3.2018'
```

	Bare	**LXC**	**Docker**
Průměr	0,50 CPU	0,51 CPU	0,18 CPU
Medián	0,07 CPU	0,18 CPU	0,13 CPU
Směrodatná odchylka	0,73 CPU	0,53 CPU	0,19 CPU

Tabulka 19 - Základní charakteristiky naměřených dat zatížení CPU se zátěží

Vysoká směrodatná odchylka odpovídá vysokému nárůstu využití CPU během testu. Vzhledem k tomu a také k předpokládanému dlouhodobému dlouhodobějšímu rozložení zátěže v reálných podmínkách je pro další analýzu využit medián naměřených dat.

Nízké využití procesoru v případě instalace s Docker kontejnery je způsobena omezeními, která jsou vynucována stejně jako v případě využití paměti RAM.



Obrázek 14 - Grafické zobrazení naměřených dat využití CPU se zátěží

Během generování zátěže byly také změřeny časy, ve kterých přišla zpět odpověď serveru. Ta byla ve všech případech bez chyby. Následující tabulka obsahuje vypočtený průměr a medián z naměřených dat získaných programem jMeter.

	Bare	**LXC**	**Docker**
Průměr	304 ms	336 ms	583 ms
Směrodatná odchylka	369 ms	242 ms	118 ms

Tabulka 20 - Doba odezvy na HTTP požadavky

5 Zhodnocení výsledků

Průměrné hodnoty z naměřených dat jsou využity pro porovnání spotřeby systémových prostředků jednotlivých instalovaných variant. Hodnoty využití místa na úložišti, využití paměti RAM a procesorového času jsou porovnány pomocí zvolených mediánů.

V rámci generování zátěže byly zjištěny průměrné doby odezvy na HTTP požadavky. V tomto případě poskytl použitý software pouze hodnoty aritmetického průměru, a proto není v tomto případě využit medián, který by byl vhodnější.

V následující tabulce jsou k dispozici vypočítané absolutní a relativní rozdíly varianty s LXC kontejnerem oproti variantě bez kontejnerů.

LXC – rozdíly proti variantě bez kontejnerů		
Medián využití HDD	+ 225,47 MiB	+ 7 %
Medián využití RAM bez zátěže	+ 24,64 MiB	+ 11 %
Medián využití RAM při zátěži	+ 9,48 MiB	+ 3 %
Medián využití CPU bez zátěže	+ 0,001 CPU	+ 6 %
Medián využití CPU při zátěži	+ 0,110 CPU	+ 161 %
Průměrná doba odezvy na HTTP požadavky	+ 32 ms	+10 %

Tabulka 21 - Absolutní a relativní rozdíly naměřených hodnot varianty LXC

Větší nároky na místo na úložišti jsou především způsobeny základní velikostí obrazu LXC kontejneru. Ten obsahuje soubory předinstalovaného operačního systému. Rozdíl 225 MiB nepovažuje autor, s ohledem na velikosti a ceny dostupných disků, za podstatný.

Vliv virtualizační vrstvy LXC je patrný v hodnotách využití RAM i CPU. V případě RAM se s rostoucí zátěží rozdíl zmenšuje, protože v takové chvíli je většina využité paměti zabrána aplikacemi poskytující zatěžované služby. V případě využití CPU při zátěži byl rozdíl znatelný. LXC kontejner způsobil 2,5x vyšší nároky na procesorový čas. Z tohoto důvodu se tento typ kontejneru nemusí hodit pro služby pod vysokou zátěží.

Průměrná doba odezvy na HTTP požadavky byla v případě webserveru v LXC kontejneru o 10 % vyšší. Nejedná se o velký rozdíl, ale i přesto může způsobit potřebu dodatečné optimalizace v případě náročnějších webových služeb.

V následující tabulce jsou k dispozici vypočítané absolutní a relativní rozdíly varianty s Docker kontejnery oproti variantě bez kontejnerů.

Docker – rozdíly proti variantě bez kontejnerů		
Medián využití HDD	+ 1462,94 MiB	+ 45 %
Medián využití RAM bez zátěže	+ 25,63 MiB	+ 12 %
Medián využití RAM při zátěži	-13,70 MiB	- 5 %
Medián využití CPU bez zátěže	+ 0,006 CPU	+ 64 %
Medián využití CPU při zátěži	+ 0,064 CPU	+ 94 %
Průměrná doba odezvy na HTTP požadavky	+ 279 ms	+ 91 %

Tabulka 22 - Absolutní a relativní rozdíly naměřených hodnot varianty Docker

Oproti variantě bez použití kontejnerů má instalovaná Docker varianta vysoké nároky na místo na úložišti. To je způsobeno, stejně jako v případě LXC kontejnerů, soubory systému v jejich instalovaných obrazech. V případě Docker instalace je tento nárůst vícenásobný díky použití více kontejnerů. Docker také udržuje během sestavování kontejneru jeho mezistavy z důvodu optimalizace rychlosti při opakovaném sestavování. V případě využití Docker kontejnerů je tedy nutné počítat se vyššími nároky na volné místo, hlavně pokud je jich využito mnoho pro malé aplikace.

Z hodnot využití RAM i CPU je překvapivý menší nárůst naměřených hodnot oproti variantě s LXC kontejnerem, respektive nižší nároky na operační paměť v případě provozu se zátěží. Tento rozdíl je způsoben omezenými systémovými prostředky, které jsou pro kontejnery dostupné. Přes toto omezení je využití CPU v zátěži skoro dvojnásobné a průměrná doba odezvy je téměř 2x horší. Tento rozdíl již může být pro uživatele webové znatelný.

6 Závěr

Hlavního cíle, vytvoření plně funkční sady serverových služeb, bylo dosaženo konfigurací softwaru na stejně výkonných virtualizovaných linuxových serverech distribuce Ubuntu. Tyto služby byly úspěšně nainstalovány a nakonfigurovány ve třech variantách: bez použití kontejnerizačních technologií a za využití kontejnerů LXC a Docker. Služby byly dostupné na veřejných IPv4 i IPv6 adresách.

Bezpečnostní úroveň instalovaných služeb byla otestována několika volně dostupnými webovými službami a klientskými aplikacemi v součinnosti s dalšími zdroji pro potvrzení výsledku.

Dílčím cílem bylo porovnání náročnosti jednotlivých instalací nastavených služeb z hlediska bezpečnosti a časové náročnosti. Tento cíl byl řešen analýzou kontejnerových technologií LXC a Docker v teoretické části. V praktické části byla řešena příprava kontejnerů. V této části byla veškerá nežádoucí komunikace na úrovni firewallu. Po celou dobu tvorby této práce nebyl zaznamenán průnik do systému, ani jiné narušení bezpečnosti.

Z výše zmíněné analýzy vyplývá, že kontejnery přináší, díky izolaci procesů od hostujícího operačního systému, vyšší úroveň bezpečnosti proti některým typům útoků – zvláště při využití neprivilegovaných kontejnerů. Umožňují také vyšší kontrolu nad procesy služeb, například omezením maximální možné využití systémových prostředků, což může chránit ostatní služby před zahlcením útokem hrubou silou. Oba typy kontejnerů umožňují migraci na jiné OS, u LXC však tato migrace nemusí být triviální.

Cenou za bezpečnost jsou vyšší časové a znalostní požadavky na zprovoznění služeb. V případě LXC je nutná konfigurace firewallu, hostujícího OS a kontrolních skupin. V případě Docker kontejnerů je nutná konfigurace OS a firewallu v menším měřítku. Značným omezením je neefektivní přístup testovaných kontejnerů při provozu pouze na IPv6.

Testované instalace za použití kontejnerů jsou oproti instalaci bez kontejnerů v obou variantách více náročné. Zvýšení požadavků na systémové prostředky je zřejmé hlavně v případě využití CPU při běhu pod zátěží. Průměrné naměřené hodnoty byly 2 – 2,5krát

vyšší než u varianty bez kontejnerů. Vyšší nároky mohou být problémem hlavně u méně výkonných serverů, protože možnosti navýšení procesorového výkonu jsou omezené.

V případě použití dostatečně výkonných serverů převažují výhody kontejnerizačních technologií, a to hlavně díky vyšší bezpečnosti a přenositelnosti. S ohledem na možnou automatizaci nasazení a správu jsou Docker kontejnery vhodnější pro produkční provoz i testování. Kontejnery LXC se hodí pro existující aplikace, které nejsou připravené na běh v Docker kontejnerech a případné škálování. Technologie Docker i LXC předpokládají přístupnost k síti Internet na protokolu IPv4.

Vzhledem k omezení kontejnerizačních technologií LXC a Docker se jeví, jako další vhodný krok, využití orchestračních nadstaveb jako je Kubernetes a OpenShift. Tyto nadstavby také řeší možné škálování a spouštění kontejnerů na několika fyzických, či virtualizovaných serverech.

7 Seznam použitých zdrojů

[1] Red Hat Enterprise Linux Blog. *The History of Containers*. [online]. [cit. 2017-11-15]. Dostupné z http://rhelblog.redhat.com/2015/08/28/the-history-of-containers/

[2] Medium. *Introduction to Containers: Concept, Pros and Cons, Orchestration, Docker, and Other Alternatives*. [online]. [cit. 2017-11-15]. Dostupné z https://medium.com/flow-ci/introduction-to-containers-concept-pros-and-cons-orchestration-docker-and-other-alternatives-9a2f1b61132c

[3] Docker *Documentation. Get started with Docker for Windows*. [online]. [cit. 2017-11-15]. Dostupné z https://docs.docker.com/docker-for-windows/

[4] Wikiknihy. *Virtualizace v Linuxu*. [online]. [cit. 2018-01-20]. Dostupné z https://cs.wikibooks.org/wiki/Virtualizace_v_Linuxu

[5] RUEST, Danielle. *Virtualizace*. Computer Press, 2010. ISBN: 978-80-251-2676-9

[6] GOASGUEN, Sébastien. Docker Cookbook: Solutions and Examples for Building Distributed Applications. O'Reilly Media, 2015. ISBN: 978-1491919712

[7] Linux Containers. *LXC – Introduction*. [online]. [cit. 2017-12-14]. Dostupné z https://linuxcontainers.org/lxc/introduction/

[8] Linux Containers Forum. *Comparing LXD vs. LXC* [online]. [cit. 2017-12-14]. Dostupné z https://discuss.linuxcontainers.org/t/comparing-lxd-vs-lxc/24

[9] Lxc/lxc. *Can‘t start new unprivileged container in Fedora 27 · Issue #1998*. [online]. [cit. 2017-12-14]. Dostupné z https://github.com/lxc/lxc/issues/1998

[10] IT-Offshore. *Centos 7 Unprivileged LXC Containers*. [online]. [cit. 2017-12-14]. Dostupné z https://it-offshore.co.uk/linux/CentOS/29-CentOS-7-unprivileged-lxc-containers

[11] Sourceforge. *Lxc*. [online]. [cit. 2017-12-14]. Dostupné z http://lxc.sourceforge.net/man/lxc.html

[12] Docker *Documentation. Get Started, Part 1: Orientation and Setup*. [online]. [cit. 2018-01-10]. Dostupné z https://docs.docker.com/get-started/

[13] NICKOLOFF, Jeff. Docker in Action. Manning Publications, 2016. ISBN: 978-1633430235

[14] Internet Engineering Task Force. *RFC 1700*. [online]. [cit. 2018-01-16]. Dostupné z http://www.ietf.org/rfc/rfc1700.txt

[15] HTTP/2. *HTTP/2 Frequently Asked Questions*. [online]. [cit. 2018-01-16]. Dostupné z https://http2.github.io/faq/

[16] W3 Techs. *Usage of site elements for websites*. [online]. [cit. 2018-01-16]. Dostupné z https://w3techs.com/technologies/overview/site_element/all

[17] W3 Techs. *Usage Statistics and Market Share of Web Servers for Websites, November 2017*. [online]. [cit. 2018-01-16]. Dostupné z https://w3techs.com/technologies/overview/web_server/all

[18] DENT, D. Kyle, Postfix: The Definitive Guide: A Secure and Easy-to-Use MTA for UNIX. O'Reilly Media, 2003. ISBN: 978-0596002121

[19] ALIBI, Mohamed a Roy, Bhaskarjyoti. Mastering CentOS 7 Linux Server. Packt Publishing, 2016. ISBN: 978-1785282393

[20] IANA. *Service Name and Transport Protocol Port Number Registry*. [online]. [cit. 2018-02-25]. Dostupné z http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

[21] DUBOIS, Paul. MySQL Cookbook: Solutions for Database Developers and Administrators. O'Reilly Media, 2014. ISBN: 978-1449374020

[22] Collectd Wiki. *Data source*. [online]. [cit. 2018-03-01]. Dostupné z https://collectd.org/wiki/index.php/Data_source

[23] RRDtool. *Rrdtool*. [online]. [cit. 2018-03-01]. Dostupné z https://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html

[24] OWASP. *Injection problem*. [online]. [cit. 2017-12-20]. Dostupné z https://www.owasp.org/index.php/Injection_problem

[25] Heartbleed Bug. [online]. [cit. 2017-12-20]. Dostupné z http://heartbleed.com/

[26] Aqua Blog. *Do Containers Provide Better Protection Against Meltdown and Spectre*? [online]. [cit. 2017-12-20]. Dostupné z https://blog.aquasec.com/do-containers-provide-better-protection-against-meltdown-and-spectre

[27] Akamai. *Global State of the Internet Security & DDoS Attack Reports*. [online]. [cit. 2018-01-15]. Dostupné z https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp

[28] Community Help *Wiki. UFW*. [online]. [cit. 2018-01-15]. Dostupné z https://help.ubuntu.com/community/UFW

[29] DistroWatch.com. [online]. [cit. 2018-01-15]. Dostupné z https://distrowatch.com/dwres.php

[30] OpenBSD manual pages. *sshd(8).* [online]. [cit. 2018-01-15]. Dostupné z https://man.openbsd.org/sshd

[31] Collectd Wiki. *Table of Plugins*. [online]. [cit. 2018-01-15]. Dostupné z https://collectd.org/wiki/index.php/Table_of_Plugins

[32] Linux manual page. *Subuid(5).* [online]. [cit. 2018-01-15]. Dostupné z http://man7.org/linux/man-pages/man5/subuid.5.html

[33] Linux manual page. *Subgid(5).* [online]. [cit. 2018-01-15]. Dostupné z http://man7.org/linux/man-pages/man5/subgid.5.html

[34] narkive. *Container with different architecture like arm on x86 [How-to]*. [online]. [cit. 2018-02-25]. Dostupné z http://lxc-users.linuxcontainers.narkive.com/f8zB2nic/container-with-different-architecture-like-arm-on-x86-how-to

[35] Cepharum.blog. *LXC host featuring IPv6 connectivity.* [online]. [cit. 2017-11-01]. Dostupné z https://blog.cepharum.de/en/post/lxc-host-featuring-ipv6-connectivity.html

[36] The Linux Kernel Archives. [online]. [cit. 2018-02-27]. Dostupné z https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

[37] Ubuntu in Launchpad. *UFW does not clean iptables setting from /etc/ufw/before.rules*. [online]. [cit. 2018-02-27]. https://bugs.launchpad.net/ubuntu/+source/ufw/+bug/881137

[38] Ubuntu Manpage. *Ufw – program for managing a netfilter firewall*. [online]. [cit. 2017-12-10]. Dostupné z http://manpages.ubuntu.com/manpages/xenial/man8/ufw.8.html

[39] BOWEN, Rich. Apache Cookbook: Solutions and Examples for Apache Administrators. O'Reilly Media, 2008. ISBN: 978-0596529949

[40] Mozilla SSL Configuration Generator. [online]. [cit. 2018-01-20]. Dostupné z https://mozilla.github.io/server-side-tls/ssl-config-generator/

[41] Apache HTTP Server Version 2.4. *mod-ssl.* [online]. [cit. 2018-01-20]. Dostupné z https://httpd.apache.org/docs/2.4/mod/mod_ssl.html

[42] Ivan Ristić. *CRIME: Information leakage attack against SSL/TLS.* [online]. [cit. 2017-11-15]. Dostupné z https://blog.ivanristic.com/2012/09/crime-information-leakage-attack-against-ssl-tls.html

[43] Postfix. *Postfix Configuration Parameters*. [online]. [cit. 2018-02-25]. Dostupné z http://www.postfix.org/postconf.5.html

[44] Container-images/postfix. *run postfix in the foreground – Issue #9.* [online]. [cit. 2018-03-01]. Dostupné z https://github.com/container-images/postfix/issues/

[45] WordPress. *Blog Tool, Publishing Platform, and CMS*. [online]. [cit. 2018-03-02]. Dostupné z https://wordpress.org/

[46] Apache Jmeter. *Apache JMeter*. [online]. [cit. 2018-03-02]. Dostupné z https://jmeter.apache.org/

8 Seznam zkratek

API	Application Programming Interface
BSD	Berkeley Software Distribution
CMS	Content Management System
CPU	Central Processing Unit
DoS	Denial of Service
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IMAP	Internet Message Access Protocol
IMAPS	Internet Message Access Protocol Secure
IP	Internet Protocol
LTS	Long Term Support
MDA	Mail Delivery Agent
MitM	Main In the Middle
MTA	Mail Transport Agent
MUA	Mail Transport Agent
NAT	Network Address Translation
OS	Operation System
PHP	PHP: Hypertext Preprocessor
POP3	Post Office Protocol version 3
POP3S	Post Office Protocol version 3 Secure
RAM	Random Access Memory
RCE	Remote code execution
RFC	Request for Comments
RRD	Round Robin Database
SMTP	Simple Mail Transfer Protocol

SMTPS	Simple Mail Transfer Protocol Secure
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UFW	Uncomplicated Firewall
XML	Extensible Markup Language

9 Přílohy

9.1 Seznam obrázků

9.2 Seznam tabulek

9.3 Další přílohy

9.3.1 Příloha 1 – DVD s konfiguračními soubory a kontejnery

DVD s konfiguračními soubory instalovaných služeb, kontejnerů LXC a Docker a souborové systémy kontejnerů se nachází v deskách této diplomové práce.