Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Využití technologie Docker pro virtualizaci vývojového prostředí PHP / Python

Dominik Víšek

Bakalářská práce
2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Dominik Víšek**
Osobní číslo:	**I15150**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Využití technologie Docker pro virtualizaci vývojového prostředí PHP / Python**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce se zabývá analýzou architektury technologie Docker a možnosti využití tohoto kontejnerového systému s cílem vytvořit praktické prostředí pro jednoduchou instalaci vývojového prostředí v operačním systému Linux.
V teoretické části student vytvoří přehled alternativních technologií k systému Docker (např.: FreeBSD Jail, WinDocks, LXC, runC, Sandboxie, VMware ThinApp). Při zpracování praktické části nesmí být nástroje a aplikační servery instalovány přímo do hostovacího operačního systému, ale musí využívat technologii Docker kontejnerů. Práce se soustředí na sestavení vlastních images pro jednotlivé potřebné části vývojového prostředí např.: php, mysql, phpunit za pomoci nástroje "docker compose". Výstup praktické části bude demonstrován s využitím alespoň čtyř kontejnerů, např. php, mysql, phpunit, mailserver a jejich vzájemnou spolupráci. Image by neměly být závislé na jejich vnitřním obsahu, ale při instalaci dojde k uživatelské volbě odkud budou čerpat data.

Rozsah grafických prací:

Rozsah pracovní zprávy: **min. 30 s.,dop. rozsah 40 s.**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Machek Zdenek, PHPUnit Essentials, 2014, Packt Publishing Ltd., ISBN: 978-17-8328-344-6 Matthias Karl, Kane Sean P., Docker: Up and Running, 2015, O'Reilly Media, Incorporated, ISBN: 978-14-9191-757-2 Mouat Addrian, Using Docker: Developing and Deploying Software with Containers, 2015, O'Reilly Media, Inc., ISBN: 978.14.-9191-591-2 Krochmalski Jaroslaw, Developing with Docker, 2016, Packt Publishing Ltd, ISBN: 978-17-8646-633-4 Turnbull James, The Docker Book: Containerization is the new virtualization, 2014, James Turnbull, ISBN: 978-09-8882-020-3 Raj Pethuru, Chelladhurai Jeeva S., Singh Vinod, Learning Docker, 2015, Packt Publishing Ltd, ISBN: 978-17-8439-193-5 Rosebrock Eric, Filson Eric, Setting up LAMP: Getting Linux, Apache, MySQL, and PHP Working Together, 2006, John Wiley & Sons, ISBN: 978-07-8215-112-1

Vedoucí bakalářské práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2017**

Termín odevzdání bakalářské práce: **12. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

Univerzita Pardubice
L.S.
Česká republika

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 20. března 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 20. 5. 2018

Dominik Víšek

PODĚKOVÁNÍ

Rád bych touto cestou vyjádřil poděkování Ing. Monice Borkovcové, Ph.D za její cenné rady a trpělivost při vedení mé bakalářské práce. Rovněž bych chtěl poděkovat Ing. Filipu Majeríkovi za připomínky, vstřícnost a pomoc při získání potřebných informací a podkladů k tvorbě bakalářské práce.

ANOTACE

Práce se zabývá analýzou architektury technologie Docker a možnosti využití tohoto kontejnerového systému s cílem vytvořit praktické prostředí pro jednoduchou instalaci vývojového prostředí pro PHP/Python v operačním systému Linux s využitím technologie Docker. V rámci práce je demonstrováno využití této technologie a zároveň je provedena analýza technologie Docker s následným porovnáním s podobnými virtualizačními nástroji. V rámci práce bude virtualizováno vývojové prostředí pro lokální využití pro vývoj webových stránek. Virtualizované vývojové prostředí bude řádně otestováno a v rámci ověření funkčnosti budou vytvořeny dvě stejné webové stránky za použití dvou různých PHP frameworků.

KLÍČOVÁ SLOVA

Docker, kontejnerový systém, vývojové prostředí, virtualizace.

TITLE

Using Docker for PHP / Python development virtualization

ANNOTATION

The thesis deals with the analysis of the Docker architecture and the possibilities of using this container system to create a practical environment for simple installation of PHP / Python development environment in the Linux operating system using Docker technology. In the course of the work, the use of this technology is demonstrated and at the same time the Docker analysis is performed with subsequent comparison with similar virtualization tools. In the framework of the thesis, the development environment for local development for the development of websites will be virtualized. A virtualized development environment will be properly tested and it would be created two validation pages using two different PHP frameworks.

KEYWORDS

Docker, container system, development environment. Virtualization, PHP

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

HTML	HyperText Markup Language
PHP	PHP: Hypertext Preprocessor
CSS	Cascading Style Sheets
MIT	Massachusetts Institute of Technology
ORM	Object-Relational Mapping
OS	Operační systém
GNU GPL	GNU General Public License
MAC	Multiple Access Computer
CGROUPS	Control Groups
AMD	Advanced Micro Devices
CLI	Command line
REST	Representational State Transfer
API	Aplication Programming Interface
ISV	Independent Software Vendors
XHTML	Extensible HyperText Markup Language
SSL	Secure Socket Layer
DB	Database

ÚVOD

Téma Bakalářské práce „Využití technologie Docker pro virtualizaci vývojového prostředí PHP / Python“ je spojeno s pojmem virtualizace, která je v současné době opět na vzestupu, a to hlavně díky nástrojům jako je právě Docker. Autorovou motivací bylo převážně poznání nové technologie, která je v praxi často využívána a zájem o virtualizovaná prostředí.

Každý webový projekt vyžaduje různé technologie, jak pro samotné spuštění aplikace, tak i pro její vývoj. Některé projekty vyžadují vynaložení velkého úsilí pro jejich zprovoznění, občas i v řádech několika dnů. K vyřešení tohoto problému lze právě využít virtualizaci vývojového prostředí. Tímto lze docílit pouhou instalaci virtualizačního nástroje, pomocí kterého by byl zprovozněn webový projekt se všemi potřebnými nástroji a technologiemi. Virtualizace vývojového prostředí však nepřináší výhody jen pro vývojáře, kteří se připojí do nových projektů, ale i pro samotný projekt.

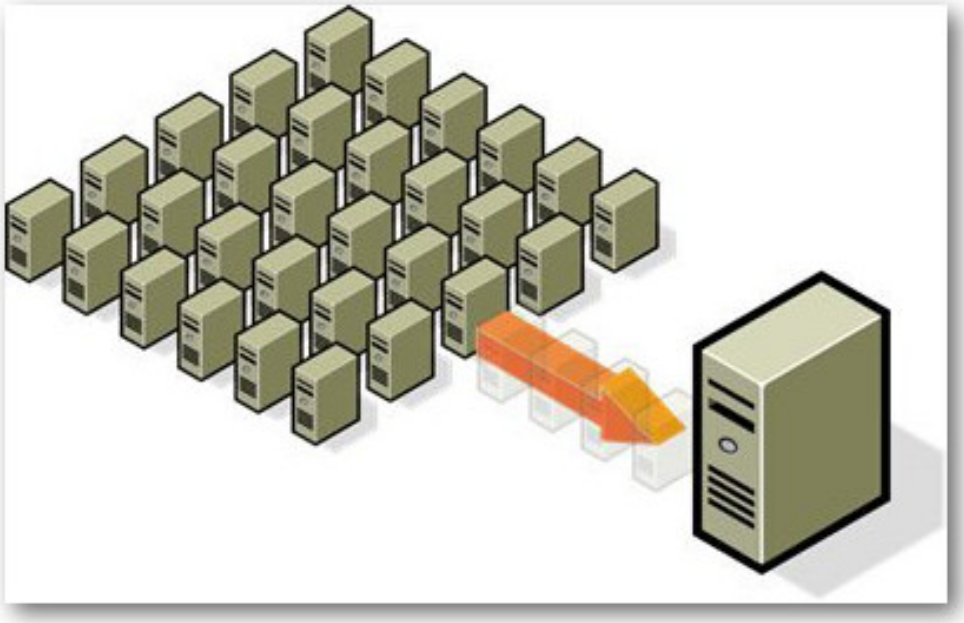
Teoretická část práce seznamuje čtenáře s obecnými pojmy jako je virtualizace a virtualizačním nástrojem Docker. U virtualizačního nástroje Docker je podrobně rozebrána struktura, využití a implementace.

Praktická část práce aplikuje teoretická východiska bakalářské práce. Výstupem praktické části bude vytvoření virtualizovaného vývojového prostředí zejména pro lokální vývoj. Vytvořené virtualizované vývojové prostředí bude demonstrováno s využitím alespoň čtyř kontejnerů, např. php, myslq, phpunit, mailserver a jejich vzájemné spolupráci. V rámci ověření správné funkčnosti vývojového prostředí budou za pomocí virtualizovaného vývojového prostředí vytvořeny dvě webové aplikace, které budou obsahovat stejný vzhled a stejnou funkčnost za použití dvou různých PHP frameworků.

Důraz výsledné aplikace bude kladen zejména na jednoduché ovládání a pohodlný vývoj webových projektů za použití virtualizovaného vývojového prostředí. Součástí praktické části bude provedení testování vývojového prostředí. Testována bude funkčnost všech částí prostředí na nejpoužívanějších operačních systémech, jako jsou Windows, Linux a Mac OS.

1 VIRTUALIZACE

Virtualizace je technologie umožňující provozovat více nezávislých virtuálních strojů na jednom skutečném fyzickém stroji. Virtualizaci lze popsat jako vytvoření zdánlivého (virtuálního) počítače uvnitř skutečného počítače za pomocí vhodného softwaru. Virtualizace tedy poskytuje určitou míru abstrakce. Technologie byla vytvořena jako reakce na problematiku vývoje a následného debuggingu aplikací. V současné době je k dispozici uživatelům mnoho různých typů virtualizace. Jednotlivé typy se od sebe odlišují různými způsoby implementace, možnostmi svého využití a samozřejmě svými výhodami a nevýhodami.[18][17]



Obrázek 1: Demonstrace virtualizace, zdroj: [36]

1.1 Historie

Historie virtualizace započala v šedesátých letech dvacátého století společností IBM, která v těchto letech nabízela širokou škálu operačních systémů. Každá nová generace jejich operačních systémů byla velice odlišná od generace přechozí. Pro zákazníky IBM to však byla poměrně zásadní komplikace, jelikož bylo problematické splňovat potřebné hardwarové požadavky pro daný operační systém. Zároveň tamní počítače zvládaly dělat pouze jednu činnost v jeden určitý okamžik. Na druhé straně nebyl tento problém společností IBM vnímán na tolik negativně. Počítače se nacházely primárně ve vědecké sféře a zdálo se, že systémy jsou pro zákazníky plně dostačující.[21]

Vedení společnosti IBM se rozhodlo, vzhledem k tomu, že již nabízejí velké množství operačních systémů, že začnou pracovat na novém systému pro sálové počítače s názvem S/360. Ten byl navržen tak, aby mohl být použit jako náhrada za většinu dosavadních operačních systémů a aby udržoval zároveň zpětnou kompatibilitu. Když byl systém poprvé navržen, mělo se jednat o jednouživatelský systém pro spuštění dávkových úloh. [21]

Vše se změnilo 1. července 1963, kdy Massachusetts Institute of Technology (MIT) oznámilo projekt *MAC*, který byl později přejmenován na *Multiple Access Computer* neboli počítač s více uživatelskými přístupy. MIT na tento projekt získalo grant a vybralo si, jako svého dodavatele pro počítačový hardware, společnost GE. [21]

Ztráta příležitosti byla pro firmu IBM určitým impulzem. V reakci na poptávku od MIT a Bell Labs, navrhlo IBM CP-40, což je operační systém, který implementoval úplnou virtualizaci. Nicméně ten na trhu zákazníkům nebyl nikdy nabídnut, a v brzké době byl nahrazen nástupcem s názvem CP-67. [21]

Z důvodu nástupu populárnějšího řešení klient-server nebyla v následujících letech virtualizaci věnována příliš velká pozornost. V 90 letech ale přichází na scénu společnost VMware, která představuje první živou migraci. Jinými slovy se jedná o přemístění virtuálního stroje, mezi fyzickými stroji, bez výpadku systému. Další velký krok v technologii virtualizace nastává až v roce 2000. V tomto roce dochází k přidání virtualizační technologie do samotných procesorů od výrobců Intel a AMD. [3]

V současné době patří pojem virtualizace k poměrně často diskutovaným tématům. Často virtualizaci spojujeme s budoucností v oblasti technologií a to právě díky technologiím jako jsou například Docker nebo VMware.

1.2 Druhy virtualizace

1.2.1 Emulace

Emulace nebo jinými slovy simulace je typ virtualizace, který se používá pro simulaci operačního systému, který není původně určen pro daný fyzický hardware. Virtualizovány jsou veškeré komponenty jiné hardwarové platformy. Vzhledem k tomu, že emulace simuluje odlišnou hardwarovou strukturu, není možné, aby byla využívána hardwarová podpora virtualizace, kterou dnešní procesory již disponují. Jedná se tedy pouze o softwarovou

virtualizaci. Proto také vyžaduje vysokou režii. Příklady softwarových nástrojů sloužících pro emulaci: PearPC, Bochs, DOSBox. [18][34]

1.2.2 Úplná virtualizace

Jedná se o běžný a efektivní typ virtualizace, která se používá ke spuštění nemodifikovaných operačních systémů stejné architektury, jako je fyzický počítač. V rámci virtualizace se vytváří kompletní virtuální hardware a emulovány jsou všechny součástky počítače. V rámci úplné virtualizace vzniká téměř ideální stav, při kterém dochází k oddělení fyzické vrstvy. Z toho vyplývá, že veškeré programy jsou spouštěny pouze v simulovaných součástkách fyzického stroje a jednotlivé přístupy k fyzickému hardwaru jsou pouze zprostředkovávány za pomocí *hypervisoru*. Úplná virtualizace má řadu výhod. Pro příklad je to možnost vytvořit si virtuální hardware dle potřebných parametrů, nezávislost programů na konkrétním technickém vybavení nebo snadné přenášení a zálohování virtualizovaných operačních systémů. Za nevýhodu je možné považovat pomalou práci se vstupně výstupními zařízeními. Nejznámějšími zástupci úplné virtualizace jsou například VirtualBox, VMware a Workstation. [28][18]

1.2.3 Virtualizace na úrovni jádra operačního systému

Virtualizace na úrovni jádra operačního systému, občas přezdívaná jako kontejnerová virtualizace, je typ, jehož koncepce je založena na využití jádra hostitelského operačního systému, několika izolovanými virtuálními stroji. Pomocí této virtualizace je možné provozovat více instancí stejných operačních systémů jako je hostující operační systém. V rámci jednoho operačního systému se vytváří vzájemně oddělená prostředí neboli kontejnery. Takto lze například jednoduše provozovat na jednom fyzickém stroji několik webových serverů, přičemž pro každý server nemusí být nainstalovaný kompletní operační systém, protože jádro hostujícího systému může přidělit a sdílet jednotlivé prostředky jednotlivým kontejnerům. Hlavní výhodou tohoto typu virtualizace je jednoduchá správa kontejnerů. Oproti tomu, operační systémy nejsou plně odděleny, mají společné jádro hostujícího operačního systému. Tento fakt je v mnoha situacích výhodou, ale i nevýhodou. Příkladem kontejnerové virtualizace je Docker a OpenVZ.[34][18][17]

1.2.4 Para-virtualizace

Jedná se o typ virtualizace, při kterém není vytvářen kompletní virtuální hardware počítače. Používá se v situacích, při kterých se některé součástky virtuálního a fyzického počítače shodují. V případě, kdy se bude shodovat procesor, i když bude mít nižší výkon, lze tedy virtualizaci tohoto typu použít. Pro komunikaci s fyzickými součástkami počítače se používá hypervizor, což je rozhraní, které slouží pro komunikaci jádra virtualizovaného operačního systému s fyzickým hardwarem. Typickým zástupcem tohoto typu virtualizace je Xen. Pomocí nástroje Xen je možné provádět paravirtualizaci operačních systémů, pokud bylo jádro daného systému patřičně modifikováno. Nicméně i tento nástroj byl vylepšen a v dnešní době je možné provádět paravirtualizaci i pro běh nemodifikovaného operačního systému. Za výhodu paravirtualizace je považován hlavně vysoký výkon, jelikož většinu výpočtů provádí přímo fyzický hardware. Nevýhodou je však nutnost instalace ovladačů na hostující operační systém a některé případy také vyžadují modifikaci jádra operačního systému. [18][28][34]

2 DOCKER

Docker je jeden z nástrojů, který se využívá při virtualizaci na úrovni jádra operačního systému pomocí takzvaných kontejnerů. Jednou z hlavních myšlenek Dockeru je spouštět aplikace nezávisle na operačních systémech a snadno tyto aplikace také přenášet mezi jednotlivými systémy.[9]

Docker je zejména využíván vývojáři aplikací. Pomocí kontejnerů je vývojářům umožněno vytvořit balíček aplikací se všemi částmi, které jsou potřeba ke spuštění aplikace. Výsledkem je tedy pouze jeden balíček, který je spustitelný na libovolném počítači, bez ohledu na to, jaké nastavení a aplikace bude obsahovat. Jedinou podmínkou je mít nainstalovanou technologii Docker.[9][15]

2.1 Technologie na které je Docker vybudován

Docker je naprogramován v programovacím jazyce *Go*. Dále také využívá funkce, které jsou poskytovány jádrem operačního systému Linux a dalších technologií, které jsou popsány v dalších kapitolách. [9]

2.1.1 Go

Go je multiparadigmatický programovací jazyk, který vytvořila společnost Google v roce 2007. Programovací jazyk vznikl jako reakce na některé problémy v oblasti vývoje softwarové infrastruktury společnosti Google. Go byl navržen tak, aby řešil problémy, kterým v tu dobu společnost čelila při vývoji softwaru. Z tohoto důvodu se nejedná o nový průlomový programovací jazyk. Go je však velkou skupinou lidí považován za vynikající nástroj pro vývoj velkých softwarových projektů. Projekty jsou vedeny pod licencí *open-source*, jedná se tedy o projekt s otevřeným zdrojovým kódem, do kterého může kdokoliv nahlížet či přispívat.[33][16]

2.1.2 Namespaces

Namespaces nebo také jmenné prostory, je technologie, kterou využívá Docker k poskytnutí izolace jednotlivým kontejnerům. Každý jednotlivý kontejner obsahuje své izolační aspekty.

Izolační aspekty kontejneru jsou například:

- pid (id procesu),
- ipc (správa sdílené paměti s ostatními kontejnery),
- net (správa síťovách rozhraní).

Každý aspekt kontejneru je spuštěn ve svém jmenném prostoru, na který je také omezen. Docker každému spouštěnému kontejneru přiřadí sadu jmenných prostorů v okamžiku jeho spouštění. Jmenné prostory používá i Docker Engine, který má různé jmenné prostory pro procesy, síťová rozhraní, izolaci jádra atd.[25][9]

2.1.3 Control Groups

Další klíčový prvek, který Docker využívá, jsou *Control groups*(kontrolní skupiny) zkráceně *cgroups*. Kontrolní skupiny sledují skupiny procesů a zároveň omezují aplikace na konkrétní soubor prostředků. Jinak řešeno dovolují Docker Enginu sdílet dostupný hardware mezi všemi běžícími kontejnery. Zároveň je možno nastavit omezení na využívání hardwaru pro jednotlivé kontejnery. Je tedy možné například omezit dostupnou paměť pro konkrétní kontejner. [9]

Kontrolní skupiny jsou součástí jádra operačního systému Linux a umožňují uspořádání procesů do hierarchicky uspořádaných skupin procesů, jejichž využití může být omezováno a sledováno. Implementace seskupování jednotlivých procesů a následné případné omezování zdrojů jsou od sebe odděleny. Implementace seskupování se vyskytuje přímo v jádru operačního systému, zatímco omezování zdrojů je implementováno pomocí subsystémů. Komunikace cgroups s aplikacemi nacházejícími se mimo jádro operačního systému je dostupná pomocí pseudo-souborového systému, který se nazývá cgroupfs.[38][13]

2.1.4 UnionFS

Union file system, přeloženo Union souborový systém, je tvořen za pomocí mnoha odlišných souborových systémů. Umožnuje seskupovat adresáře a soubory ve vrstvách. Docker obraz je tvořen pomocí více souborových systémů, které jsou vzájemně vrstvené. Vždy je jeden souborový systém zaváděcí a podobá se typickému zaváděcímu souborovému systému, který využívá Linux. Se zaváděcím souborovým systémem však uživatelé Dockeru s největší pravděpodobností nikdy nepřijdou do styku.[20][9]

V UnionFS je umožněno transparentní překrývání jednotlivých souborových systémů, čímž je docíleno, že se celý souborový systém jeví jako jeden ucelený souborový systém. Jednotlivé souborové systémy, které se nacházejí v UnionFS se nazývají *branches* neboli větve. Může nastat situace, ve které větve obsahují soubor se stejným názvem. Tento problém je řešen pomocí priority. Každá větev má svou vlastní prioritu, takže pokud vznikne konflikt souborů z důvodu duplicity názvu souboru, přednost má vždy ten soubor, který má vyšší prioritu.[20][45]

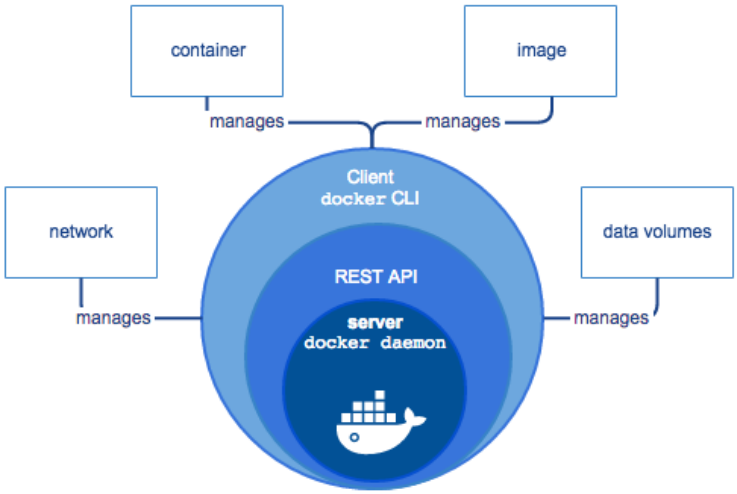
Jednotlivé větve v souborovém systému UnionFS mohou být pouze pro čtení nebo zápis. Při provedení zápisu jsou tedy všechny požadavky směrovány na konkrétní souborový systém, který je vždy určen pouze ke čtení nebo zápisu.[20]

2.1.5 Formát kontejnerů

Formát kontejnerů se skládá ze jmenných prostorů, kontrolních skupin a souborového systému UnionFS. Všechny zmíněné části jsou kombinovány a zabaleny do kontejneru pomocí Docker Engine. Docker využívá vlastní formát kontejneru, který se nazývá *libcontainer*. Ten poskytuje implementaci pro vytváření kontejnerů a všech potřebných částí. [9]

2.2 Docker Engine

Docker Engine je aplikace typu klient-server, která se skládá z více jednotlivých komponent. Mezi zmíněné komponenty patří server Docker démon, REST API, klient CLI a připojované objekty. Docker engine se používá pro úkoly, které se zabývají tvorbou, odesíláním a provozováním aplikací založených na kontejnerech. Docker Engine byl zpočátku vyvinut pouze pro operační systémy Linux, ale postupem času byl rozšířen i na další systémy. Příkladem jsou systémy Windows a mac OS. Pokud Docker Engine není pro uživatele dostačující, je možné k němu připojit zásuvné moduly. Ty jsou k dispozici v soukromých registrech nebo ve veřejných repositářích jako jsou GitHub nebo DockerHub. [9][39]



Obrázek 2: Docker Engine, zdroj: [9]

2.2.1 Docker démon

Docker démon je proces, který je spuštěn na hostitelském operačním systému. Démon čeká na požadavky, které jsou odesílány od Docker API. Po obdržení příchozího požadavku je vyvolána patřičná reakce dle typu daného požadavku. Úkolem démona je na základě požadavků spravovat Docker obrazy, kontejnery, svazky a sítě. Zároveň je umožněna komunikace mezi Docker démonem a ostatními démony. Mezi účely ostatních démonů patří zejména správa jiných Docker služeb. [14][9]

2.2.2 Docker klient

Docker klient je primární způsob, který umožnuje uživateli komunikovat s Dockerem. Způsobů komunikace mezi uživatelem a Dockerem je více. Nejčastěji využívaný způsob komunikace je pomocí Docker klienta. Komunikuje se na základě příkazů, které začínají klíčovým slovem „docker“. Příkaz je následně odeslán démonu, který na daný požadavek zareaguje. Zadávané příkazy budou podrobněji rozebrány v kapitole 2.4.1. [9]

2.2.3 Docker repositáře

Docker repositáře jsou servery, které slouží k uchovávání Docker obrazů. Repositáře jsou veřejné a díky nim je uživatelům umožněno Docker obrazy koupit, prodat a některé volně

užívat. Nejznámější repositáře pro Docker jsou Docker Cloud a Docker Hub. Docker Hub je nastaven jako výchozí repositář, nicméně existuje více repositářů a výchozí repositář je možné v případě potřeby přenastavit.[10][9]

Stažení obrazu na lokální uložiště se provádí příkazem „docker pull“. Pokud je již obraz stažen, je možné jej spustit jako kontejner nebo ho využít k tvorbě vlastního obrazu pomocí Dockerfile souboru. [10][9]

2.2.4 Docker objekty

Docker objekty, jsou takové objekty, které tato technologie vytváří a se kterými pracuje, jedná se například o obrazy, kontejnery a pluginy. [9]

Jedním ze základních objektů jsou obrazy. Obrazy jsou šablony, ze kterých lze pouze číst. Obsahují informace sloužící pro vytvoření kontejneru. Obraz je jinými slovy aplikace, kterou daný uživatel chce spustit. [9]

Docker dále poskytuje možnost tvorby vlastních obrazů pomocí souboru, který se nazývá „Dockerfile“. Dockerfile je textový dokument, který obsahuje příkazy nebo také jinými slovy instrukce pro vytvoření obrazu. Dockerfile poskytuje nejen možnost tvorby svých vlastních obrazů, ale dovoluje doplnit, či modifikovat již existující obrazy. Možnost rozšíření obrazů se provádí za použití příkazu FROM. Další příkazy, které lze použít v Dockerfile souboru jsou popsány v následujícím seznamu, který čerpá ze zdrojů: [1], [12], [11], [15]. [1][12]

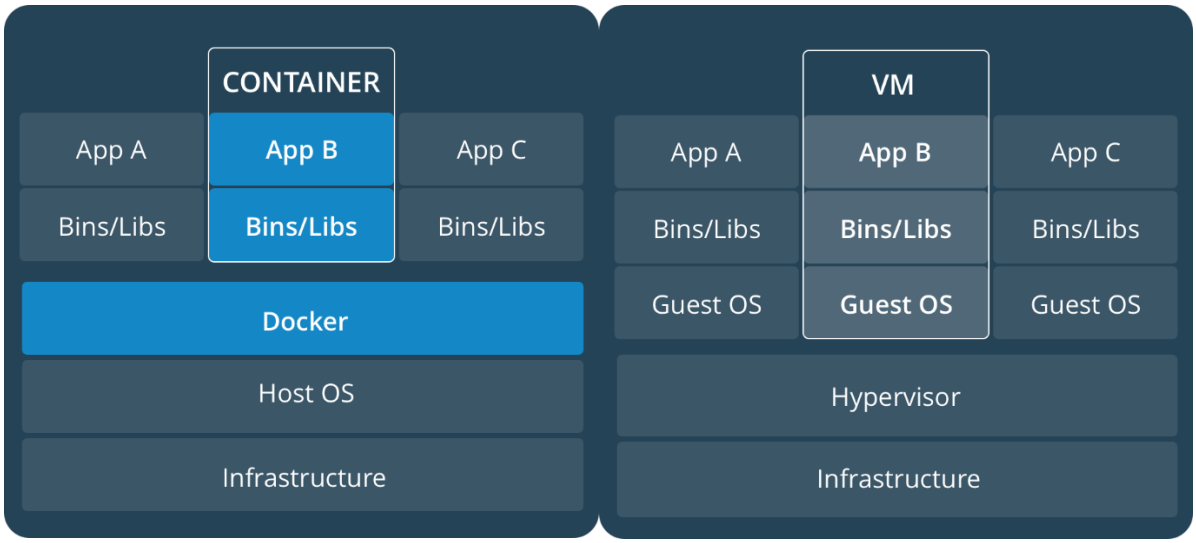
- **FROM -** Příkaz definuje základní obraz, který se má použít k zahájení procesu sestavení. Může to být jakýkoliv obraz, včetně vlastních vytvořených obrazů. Pokud se na hostiteli nenachází obraz FROM, pokusí se ho Docker najít (a stáhnout) z Docker Hubu nebo jiného úložiště obrazů. Musí to být první příkaz deklarovaný uvnitř souboru Dockerfile.
- **LABEL -** Příkaz slouží k přiřazení štítku k obrazu, pomáhá organizovat větší obrazy a zároveň uchovávat potřebné informace jako je například typ licence.
- **RUN -** Příkaz slouží k volání příkazů příkazové řádky, přičemž příkazy se provádí při sestavování kontejneru. Často se používá pro instalaci programových balíčků.
- **CMD** - Příkaz slouží k nastavení výchozího příkazu a parametrů, které je možno přenastavit, při spuštění Docker kontejneru v příkazovém řádku.

- **EXPOSE** - Příkaz slouží k přiřazení specifikovaného portu pro umožnění vytváření sítí mezi běžícím procesem uvnitř kontejneru a vnějším světem.
- **ENV -** Příkaz slouží k nastavení proměnných hodnot.
- **ADD/COPY -** Příkazy kopírují soubory ze zdrojové adresy na cílovou adresu. Příkazy ADD a COPY jsou si podobné. Nicméně se doporučuje používat příkaz COPY. Funkce COPY podporuje totiž čistě kopírování souborů, zatímco ADD podporuje jen některé funkce, které nejsou okamžitě zřejmé. Příkladem je extrakce souborů z formátu tar.
- **ENTRYPOINT -** ENTRYPOINT umožnuje konfigurovat kontejner. Příkaz je velice podobný příkazu CMD, a to jak funkčností, tak i syntaxí. Hlavní rozdíl je ve vnímání parametrů. Parametry, které jsou zadány přímo v příkazu ENTRYPOINT nebudou ignorovány ani v případě, že budou parametry zadány v příkazové řádce.
- **VOLUME** - Data volume, neboli datový svazek je speciálně určený adresář v jednom nebo více kontejnerech, který obchází Union File System. Datové svazky poskytují několik užitečných funkcí pro trvalé nebo sdílené údaje. Například přímé provádění změn dat. Data přetrvávají, i když je kontejner vymazán a existuje i možnost sdílet data mezi jednotlivými kontejnery. Samotný příkaz VOLUME je určen pro změnu cesty k datovému svazku. Je tedy možné různé adresáře seskupit do jednoho, i když se nacházejí na různých místech.
- **USER** - Příkaz USER slouží k nastavení uživatele a skupiny, pod kterým se bude spouštět Docker obraz a i následné příkazy CMD, RUN, ENTRYPOINT. Uživatele je možné nastavit pomocí názvu uživatele a jeho skupiny nebo pomocí PID a GID. Pokud uživatel nemá nastavenou primární skupinu, jako jeho skupina bude vybrána skupina root. Pokud v systému Windows neexistuje vestavěný uživatel, musí být uživatel nejprve vytvořen pomocí příkazu „net user“, který je volán v Dockerfile souboru pomocí příkazu RUN.
- **WORKDIR** - Příkaz slouží k nastavení pracovního adresáře pro všechny prováděné příkazy. Pokud nastavený adresář neexistuje, bude automaticky vytvořen. Příkaz je možné použít vícekrát v jednom Dockefile souboru. Pro každé další nastavení adresářové cesty se využívají pravidla absolutní a relativní cesty. Relativní cesta je brána od předchozí nastavené cesty k pracovnímu adresáři.

- **ONBUILD** - Aplikace může být tvořena za pomocí více Dockerfile souborů. ONBUILD slouží k odložení provedení příkazu. Používá se zejména v hlavním Dockerfile souboru a slouží k registraci pokročilých instrukcí k pozdějšímu spuštění v další části sestavení za příkazem FROM.

Dalším objektem je kontejner. Kontejner je běžící instance obrazu, z toho tedy vyplývá, že obraz je „předpis“ pro budoucí kontejner a na základě definovaného chování se kontejner z dané obrazu vytvoří. Instancí jednoho obrazu může být více, omezení je kladeno pouze na hardwarové možnosti počítače. Jednotlivé kontejnery jsou od sebe navzájem izolovány, vytváří tedy izolovaná prostředí pro spouštění aplikací. Kontejnery je možné vytvářet, spouštět, zastavovat, přesouvat nebo mazat pomocí Docker API nebo CLI. Dále je možné k nim připojit uložiště nebo internetové sítě. [43]

Důležité je znát rozdíly mezi virtuálním strojem a kontejnerem. Virtuální stroje a kontejnery jsou si velice podobné, mají podobné výhody izolace, a dokonce i přidělování zdrojů. Jejich fungování je však odlišné.[9]



Obrázek 3: Srovnání kontejneru a virtuálního stroje, zdroj: [19]

O kontejneru lze přemýšlet jako o jiné formě virtualizace. Virtuální stroje dokáží rozdělit kus hardwaru na více různých virtuálních strojů, ve kterých je možné spouštět různé operační systémy. Každý virtuální stroj obsahuje plnou kopii operačního systému zabírající velké množství paměti. Oproti tomu kontejnery virtualizují operační systém. Rozdělují ho do virtualizovaných oddílů a spouštějí kontejnerové aplikace. Právě díky tomu, že kontejnery neobsahují plnou kopii operačního systému, zabírají mnohem menší prostor

v paměti, jsou přenositelnější a efektivnější. Kontejner je tvořen kódem a potřebnými závislostmi.[19]

2.3 Docker Compose

Docker Compose je nástroj pro definování a provozování více Docker kontejnerů. Konfigurace kontejnerů je definována pomocí *YAML* souboru, který je vždy pojmenován jako „docker-compose.yml“. Díky popisovanému nástroji je po správné konfiguraci všech kontejnerů v „docker-compose.yml“ možné spustit všechny definované kontejnery za pomoci jediného příkazu „docker-compose up“. Docker compose neumožňuje pouze spouštění více kontejnerů zároveň, v souboru je dále možné definovat komunikaci či sdílet prostředky mezi jednotlivými izolovanými kontejnery.[8]

2.3.1 Konfigurační soubor docker-compose.yml

Konfigurační soubor „docker-compose.yml“ je konfigurační soubor, kterým se řídí technologie *Docker Compose*. V konfiguračním souboru je možné definovat jednotlivé kontejnery a zároveň je dle potřeby nakonfigurovat. Pro uvedení příkladu je zde možné nastavit komunikaci mezi jednotlivými kontejnery, či jim nastavit veřejné porty, na kterých mají být služby dostupné. Formátování a příkazy v konfiguračním souboru se postupně vylepšují, a proto se na začátku konfiguračního souboru uvádí *version*. Tento parametr nám udává, jaká je v konfiguračním souboru použita verze formátování, protože s každou verzí se nemění jen formátování, ale mění se i samotné atributy. Po tomto parametru již následuje definování jednotlivých služeb, které spadají v hierarchii pod klíčové slovo „services“.

```
version: '3.3'

services:
  server:
    build: ./images/apache-php
    image: bc_visek_server
    container_name: bc_visek_server
    working_dir: /server/application
```

Obrázek 4: Ukázka docker-compose.yml, zdroj: vlastní zpracování

Na výše uvedeném obrázku, je možné vidět ukázku konfiguračního souboru, který využívá formátování verze 3.3. Dále je definována Služba *server*, která se vytvoří ze souboru Dockerfile. Soubor Dockerfile pro službu server se nachází v adresáři dle cesty uvedené u parametru „build“. Vytvořenému obrazu bude nastaveno jméno „bc_docker_visek“ a vytvořený kontejner se bude jmenovat dle parametru „container_name“. Poslední parametr „working_dir“ funguje stejně jako stejnojmenný parametr v souboru Dockerfile. Příkaz slouží k nastavení výchozího adresáře pro všechny prováděné příkazy. Existuje však větší množství parametrů, které je možné využít. Ty nejpoužívanější jsou popsány v následujícím seznamu, který čerpá ze zdrojů: [8].

- **build -** Atribut definuje cestu k Dockerfile souboru ze kterého se sestavuje Docker Obraz.
- **image** - Atribut slouží k definování obrazu, ze kterého se vytváří kontejner. Pokud se daný kontejner nenachází na lokálním stroji, je stáhnut z Docker hubu, za podmínky, že je daný dostupný. Pokud je tento atribut použit v kombinaci s atributem build, atribut definuje název vytvořeného Docker obrazu.
- **working_dir** - Atribut nastavuje pracovní adresář pro definovanou službu. Jinými slovy nastavuje výchozí adresář pro spouštěné příkazy v rámci vytvořeného kontejneru.
- **container_name** - Atribut nastavuje název kontejneru pro danou službu.
- **ports** - Atribut, který slouží k mapování vnitřních portů kontejneru na takzvané „veřejné“ porty kontejneru.
- **volumes** - Atribut funguje na stejném principu jako atribut „VOLUME“, který se využívá v souboru Dockerfile. Používá se hlavně k připojení cest adresářů do jednoho svazku.
- **links** - Atribut definuje odkazy na kontejnery v jiné službě. Je možné kontejnerům přiřadit i aliasy.
- **command** - Atribut přepisuje výchozí příkaz, který mohl být nastaven například v Dockerfile souboru.

2.4 Příkazy

2.4.1 Docker

Kapitola čerpá ze zdrojů: [7]. Docker je ovládán pomocí Docker příkazů. Pro výpis všech příkazů do konzole je nutné napsat následující příkaz.

```
docker
```

Výše zmíněný příkaz neslouží pouze jako příkaz pro vypsání jiných příkazů, ale vyskytuje se také ve všech dostupných Docker příkazech. Příkazů se vyskytuje v Dockeru obrovské množství, z tohoto důvodu v této kapitole budou popsány jen ty nejdůležitější. Základní rozdělení příkazů je na normální příkazy a příkazy pro správu.

Normální příkazy jsou takové příkazy, pomocí kterých je tato technologie ovládána. Základní příkazy jsou uvedeny v tabulce **Tabulka 1**.

Příkaz	Účel příkazu
docker run	Spuštění kontejneru z obrazu.
docker stop	Zastavení kontejneru.
docker build	Sestavení obrazu ze souboru Dockerfile.
docker rm	Odstranění kontejneru.
docker rmi	Odstranění obrazu.
docker ps	Vypsání kontejnerů.
docker pull	Stažení obrazu nebo zdrojových kódů z repositáře.
docker push	Nahrání obrazu nebo zdrojových kódů do repositáře.
docker kill	Vynucené končení kontejneru.
docekr restart	Restartování kontejneru.
docker rename	Přejmenování kontejneru.
docker pause	Pozastavení běžícího kontejneru.

Tabulka 1: Přehled nejpoužívanějších základních Docker příkazů

Další kategorií příkazů jsou příkazy pro správu. Tyto příkazy slouží ke správě Docker objektů jako jsou kontejnery, obrazy nebo pluginy. Příkazy jsou uvedeny v tabulce **Tabulka 2**.

Příkaz	Účel příkazu
docker checkpoint	Správa kontrolních bodů.
docker config	Správa konfiguračních souborů.
docker container	Správa kontejnerů.
docker image	Správa obrazů.
docker network	Správa sítí.
docker plugin	Správa doplňků.
docker service	Správa služeb.
docker system	Správa systému.
docker volume	Správa svazků.

Tabulka 2: Přehled nejpoužívanějších Docker příkazů pro správu

Dá se říct, že tyto příkazy slouží pouze jako přístupový bod k dalším příkazům, které pracují s konkrétními objekty. Pokud by bylo potřeba spravovat obrazy, tak stačí napsat následující příkaz, který vypíše příkazy, které je možné použít.

```
docker image
```

Pro uvedení příkladu takto vypadá příkaz pro vypsání všech Docker obrazů.

```
docker image ls
```

2.4.2 Docker Compose

Docker compose je stejně jako Docker ovládán pomocí příkazů v příkazové řádce. Hlavní příkaz pro ovládání této technologie je následující. [31]

```
docker-compose
```

Tento příkaz funguje na stejném principu jako příkaz „docker“. Příkaz vypíše všechny možné použitelné příkazy a je také obsažen ve všech dostupných Docker Compose příkazech. Přehled základních příkazu je možné nalézt v tabulce **Tabulka3**. [31]

Příkaz	Účel příkazu
docker-compose build	Sestavení služeb.
docker-compose restart	Restartování služeb.
docker-compose config	Validace, zobrazení obsahu z docker-compose.yml.
docker-compose ps	Zobrazení Docker kontejnerů.
docker-compose up	Vytvoření a spuštění definovaných Docker kontejnerů.
docker-compose down	Ukončení a odstranění definovaných Docker kontejnerů.
docker-compose kill	Vynucené ukončení.
docker-compose start	Spuštění služeb.

Tabulka 3: Přehled nejpoužívanějších Docker Compose příkazů

3 ALTERNATIVNÍ TECHNOLOGIE

3.1 Vagrant

„Vagrant je nástroj, který vytváří kompletní přenosné vývojové prostředí pomocí virtualizace. V podstatě vytvoří na našem stroji další virtuální počítač, který muže být konfigurován například stejně jako produkční linuxový server.“ [40][44][26]

Vagrant slouží jako nástroj pro spojení technologií pro virtualizaci s nástroji pro konfiguraci serverů. Vagrant tedy nemůže fungovat samostatně, ale musí být nainstalován spolu s nějakým virtualizačním nástrojem, jako je VirtualBox. Jde tedy jinými slovy o nástroj pro zajištění konzistentního stavu vývojového prostředí při práci s více operačními systémy. [40][44]

Konfigurační soubor této technologie se nazývá Vagrantfile. Sámotný kód v konfiguračním souboru se píše v programovacím jazyce Ruby. V konfiguračním souboru je možné nastavit jaký Virtuální stroj má vagrant spustit a případné další potřebné parametry. [40] [26]

3.2 FreeBSD Jail

FreeBSD Jail je implementací virtualizace na úrovni jádra operačního systému, která umožňuje správcům systému rozdělit počítačový systém, založený na FreeBSD, na několik nezávislých mini systémů nazvaných „jails“. [27]

Tato implementace vznikla na základě potřeby sdíleného prostředí hostingových služeb, aby bylo možné vytvořit jasné oddělení vlastních hostingových služeb od služeb jednotlivých zákazníků, zejména z důvodu bezpečnosti a správy systému. [27]

3.3 WinDocks

WinDocks je otevřený virtualizační nástroj který byl zveřejněn roku 2016 a obsahoval Docker engine navržený pro provoz na Windows Serveru s podporou Net a SQL Serveru v kontejnerech. WinDocks používá Docker Engine pro přístup k Docker démonovi spolu s Windows kontejnery. Díky tomu, že WinDocks využívá Docker API je zde možná interakce s Dockerem, jako je například používání Docker příkazů. Cílem nástroje je modernizace Windows SQL serveru za pomocí virtualizace. [48] [47]

4 TECHNICKÉ ZÁZEMÍ

Pro zprovoznění projektu je nutné mít nainstalovaný virtualizační nástroje Docker a Docker Compose.

4.1 Instalace virtualizačního nástroje Docker

Virtualizační nástroj Docker je dostupný v komunitní a enterprise edici. Enterprise edice je pak dále rozdělena dle dostupných funkcí, které se s danou edicí instalují. Komunitní edice je základní edice, která poskytuje pouze základní funkce a možnosti jako Docker Engine, sestavování kontejnerů, vytváření sítí a zabezpečení. Oproti tomu, plná Enterprise edice navíc obsahuje certifikovanou infrastrukturu, rozšiřující pluginy a ISV kontejnery, správu obrazů, správu aplikací v kontejnerech a kontrolu zabezpečení obrazů. Výběr edice je závislý na individuální volbě a na potřebných funkcích. Samotní vývojáři jsou toho názoru, že komunitní verze je vhodná pro malé týmy, které začínají s technologií Docker. Jinými slovy je vhodná hlavně pro osvojení této technologie a pro experimentování s ní, proto také tato edice obsahuje dva aktualizační kanály „stable“ a „edge“. Aktualizační kanály jsou dva, protože stable verze poskytuje nové aktualizace každé čtvrtletí, přičemž aktualizace jsou ověřené a otestované, zatímco edge přichází s aktualizacemi každý měsíc a aktualizace obsahuje vždy nové funkce. Enterprise edice má oproti komunitní edici pouze jeden aktualizační kanál, kde jsou aktualizace zveřejňovány dvakrát za jeden rok.[1][37]

Docker je možné nainstalovat na všechny dostupné operační systémy, jako jsou Mac OS, Linux a Windows.[22][23][24]

4.1.1 Instalace virtualizačního nástroje Docker na operační systém Ubuntu

Kapitola čerpá ze zdrojů: [22]. Instalace má několik podmínek. Je vyžadována 64 bitová verze systému a dále instalovaná verze Ubuntu, musí být verze 14.04(LTS) a vyšší.

Pokud již existuje starší nainstalovaná Docker verze, je nutné tuto verzi před instalací odinstalovat následujícím příkazem:

```
sudo apt remove docker docker-engine docker.io
```

Tento příkaz je vhodné spustit vždy před instalací nástroje Docker. V případě, že balíčkovací systém nenalezne nainstalovaný balíček virtualizačního nástroje Docker, pouze se vypíše zpráva, že daný balíček není nainstalován.

Po případné odinstalaci starších verzí je již možné nainstalovat verzi novou. Nejdříve je však nutné nastavit správný repositář. To se provede v následujících čtyřech krocích.

1. Aktualizace nainstalovaných balíčků pomocí balíčkovacím systému APT. Příkaz však nestahuje nové verze balíčků, pouze aktualizuje seznam na nové verze balíčků. Pro instalovaní nových balíčků je nutné použít příkaz upgrade.

```
sudo apt-get update
```

2. Instalace potřebných balíčků pro přístup k Docker repositáři přes HTTPS.

```
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

3. Přidání oficiálního GPG klíče pro Docker.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Po přidání klíče je možné ověřit, zda tento klíč existuje s patřičným otiskem. To se provede pomocí následujícího příkazu. Příkaz vyhledá klíč na základě otisku, to je provedeno vyhledáním posledních 8 znaků z otisku.

```
sudo apt-key fingerprint <poslednich_8_znaku_z_otisku>
```

4. Nastavení „stable“ repositáře.

```
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

K instalaci se používá balíčkovací systém APT, proto je syntaxe stejná jako při instalaci jakéhokoliv jiného balíčku. Nedříve je nutné aktualizovat seznam balíčků a pak je možné

bezpečně instalovat, jak je demonstrováno v příkazech níže. Zde je uveden příklad instalace komunitní verze nástroje Docker.

```
sudo apt-get update

sudo apt-get install docker-ce
```

4.1.2 Instalace na operační systémy MacOS a Windows

Kapitola čerpá ze zdrojů: [23],[24]. Pro instalaci je nutné si stáhnout patřičnou verzi z oficiálních stránek Dockeru, která je ve formátu „.dmg“ pro Mac OS a „exe“ pro Windows. Stačí pak pouze pro instalaci patřičný soubor spustit. Pro instalaci však existují určité podmínky jako například minimální podporovaná verze systému. V aktuální verzi Dockeru je nutné mít nainstalovaný operační systém Windows 10 a Mac OS musí mít verzi systému 10.11 a novější.

Pokud systémy nesplňují požadavky pro spuštění Dockeru, je možné si nainstalovat takzvaný *Docker Toolbox*. Docker toolbox je nástroj, který slouží k instalovaní a spuštění prostředí dockeru na starších operačních systémech, které nesplňují potřebné minimální požadavky. Součástí Docker Toolbox je *Docker*, *Docker Compose* a *Docker Machine*.

Další částí, která je nezbytná pro běh a je také součástí toolbox balíčku je virtual box VM, který se v Docker Toolbox nazývá default. Docker Toolbox spouští „boot2docker“ linuxovou distribuci, ve které je umístěn Docker-engine s patřičnými certifikáty umístěnými na lokálním počítači. Linuxová distribuce je v balíčku obsažena z důvodu doplnění potřebných funkcí linuxového jádra.

5 VÝVOJ SOFTWARU

Vývoj softwaru by měl probíhat v několika fázích od počáteční analýzy až po finální testování (**Příloha A** – *Diagram vývoje softwaru*). Fáze jsou postupně popsány v následujících podkapitolách.

5.1 Analýza požadavků

Analýza požadavků je první fáze vývoje jakéhokoliv softwaru. Cílem analýzy požadavků je získání popisu funkčnosti daného softwaru. Kromě získání popisu funkčnosti je dále možné odhadnou čas tvorby daného projektu či jeho cenové ohodnocení. Analýza v této kapitole se zaměřuje na tvorbu virtualizovaného vývojového prostředí a je tvořena funkčními a nefunkčními požadavky.[49]

5.1.1 Funkční požadavky

- FR01 Zobrazení webových stránek

  Virtualizované vývojové prostředí umožní spouštět webové stránky.

- FR02 Správa databáze

  Virtualizované vývojové prostředí umožní přístup k databázi za pomocí nástroje Adminer.

- FR03 Spouštění testů

  Virtualizované vývojové prostředí umožní spouštět phpunit testy.

- FR04 Composer

  Virtualizované vývojové prostředí umožní spravovat závislosti za pomocí nástroje Composer.

- FR05 Nové weby

  Virtualizované vývojové prostředí umožní přidat neomezené množství webů s nastavenými virtuálními hosty.

5.1.2 Nefunkční požadavky

- NFR01 Docker virtualizace

  Vývojové prostředí bude virtualizováno za pomocí nástroje Docker.

- NFR02 Ovládání

  Virtualizované vývojové prostředí bude ovládáno za pomocí příkazové řádky.

- NFR03 Rozšiřitelnost

Virtualizované vývojové prostředí bude navrženo tak, aby mohlo být jednoduše rozšířeno o další potřebné technologie.

- NFR04 Univerzálnost

  Virtualizované vývojové prostředí bude navrženo pro snadnou přenositelnost mezi jinými platformami za použití technologie Docker.

5.2 Použité nástroje k tvorbě softwaru

5.2.1 Vývojové prostředí

V rámci projektu musí být virtualizováno vývojové prostředí pro tvorbu webů. Vývojové prostředí musí obsahovat základní nástroje pro vývoj určitého typu aplikací. V rámci této praktické části je zapotřebí vytvořit vývojové prostředí pro PHP aplikace, které využívají Framework Symfony nebo Nette. Pro zprovoznění aplikací výše zmíněného typu je nutné, aby virtualizované prostředí obsahovalo PHP, Apache a databázi. Dále je zapotřebí, aby vývojové prostředí obsahovalo nástroje Adminer a Composer, jejichž opodstatnění bude vysvětleno níže.

PHP, celým názvem PHP: Hypertext Preprocessor je nejrozšířenější serverový skriptovací programovací jazyk pro tvorbu webových stránek. Původní název pro PHP byl Personal Home Page. Název však byl později změněn, aby odpovídal rekurzivním zkratkám z projektu GNU. PHP bylo vytvořeno v roce 1994, původně jako práce pouze jednoho člověka jménem Rasmus Lerdorf. K projektu se později připojilo více lidí, a tak postupně vznikal nejrozšířenější serverový skriptovací jazyk. Všechny PHP skripty jsou prováděny na straně serveru, který vrací výsledek jednotlivých skriptů. Za pomocí PHP jsou tvořeny dynamické webové stránky a k zobrazování využívají značkovací jazyky jako HTML, XHTML, které lze použít v PHP souborech. Pro kvalitnější vývoj za pomocí PHP bylo sepsáno i několik Frameworků jako například Nette a Symfony. [46]

Apache http server je open-source http server pro operační systémy Windows a Unixové operační systémy. Projekt byl spuštěn v roce 1995 a již od roku 1996 se jedná o nejpopulárnější webový server. Apache poskytuje rozšiřitelný server, který poskytuje služby http synchronizované s aktuálními standardy.[35]

Protokol http je z pohledu OSI/ISO aplikační protokol, který funguje na základě požadavků a odpovědí. Protokol http není bezpečný, a proto je v praxi využíván protokol https. Protokol https je variantou protokolu http, který je provozován na zabezpečené vrstvě SSL. SSL je vrstva mezi transportní a aplikační vrstvou, která poskytuje zabezpečení komunikace šifrováním a autentizací. [35]

MariaDB je relační databázový systém, který vznikl z databázového systému MySQL, kvůli udržení licence svobodného softwaru GNU GPL.[41]

Adminer je nástroj pro správu databáze. Je implementován pomocí jazyka PHP. Tvořen je pouze jedním PHP souborem a umožňuje přehledně a jednoduše spravovat různé databáze jako: MySQL, MariaDB, PostgreSQL, SQLite, MS SQL, Oracle, Firebird, SimpleDB, Elasticsearch a MongoDB.[6]

Composer je nástroj pro správu knihoven psaných v jazyce PHP. Composer tedy umožňuje instalovat a spravovat PHP knihovny pomocí příkazů. Nejedná se však o balíčkovací systém jako je třeba APT. Sice spravuje balíčky a knihovny, ale jen v rámci jednoho projektu. Instalované balíčky a závislosti jsou deklarovány v souboru „composer.json“ a pokud není definováno jinak, knihovny jsou stahovány do složky *vendor*. Vendor se nachází v každém projektu, kde je Composer využit. Pokud definovaná složka neexistuje, je při instalaci balíčků za pomocí Composeru automaticky vytvořena. Composer je nezbytná záležitost, pokud se jedná o vývoj ve Frameworku jako je Nette nebo Symfony, protože knihovny těchto Frameworků jsou taktéž instalovány za pomocí Composeru.[32]

5.2.2 Technologie pro tvorbu webových stránek

Základním kamenem vytvářených projektů jsou PHP Frameworky. Framework je nadstavba programovacího jazyku, který má za cíl řešit nejčastější případy užití a často implementované požadavky. Lze konstatovat, že se těmito nástroji usnadňuje samotný vývoj aplikace, jelikož je již vývojář nemusí znovu implementovat. Mezi nejznámější PHP Frameworky patří Nette Framework a Symfony Framework.

Nette Framework je PHP Framework, který je vyvíjen českými vývojáři. Využívá ve výchozí konfiguraci šablonovací systém *latte* pro vykreslování stránky za pomocí HTML a pomocných maker. Jednotlivé stránky jsou tvořeny pomocí *render metod* v *presenteru* a také přidělených *latte* šablon. Nette využívá pomocných knihoven, které jsou instalovány pomocí Composeru.

Symfony Framework využívá stejně jako Nette Framework Composer. Výchozí šablonovací systém, je však rozdílný a nazývá se *twig*. Ačkoliv se jedná o dva různé Frameworky, v některých věcech jsou si hodně podobné. Jednotlivé stránky jsou tvořeny pomocí metod nejčastěji nazvaných *action* a umístěny jsou v *Controlleru*.

HTML je textový značkovací jazyk, celým názvem HyperText Markup Language. Jazyk HTML je využit téměř na každé webové stránce, a to zejména pro definování významu obsahu stránek. Jazyk je využíván v kombinaci s CSS, který je popsán níže.[4]

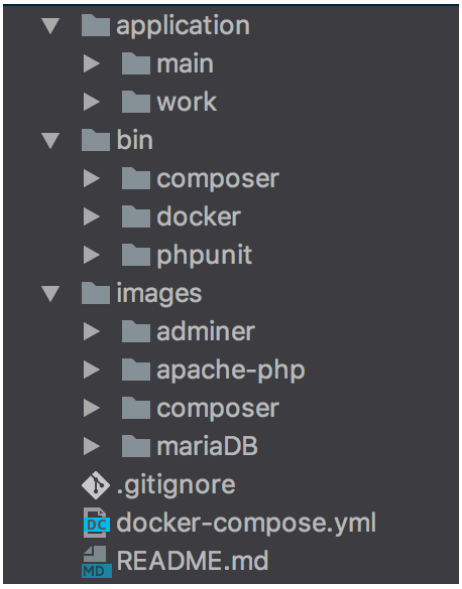
Jazyk vznikl v devadesátých letech minulého století. Na začátku popisoval pouze základní elementy, jako jsou tabulky či seznamy pro tvorbu jednoduchých webových stránek. Jazyk se od té doby vyvinul do podoby s názvem HTML5, ten bude využit v rámci tvorby ukázkových webových stránek. Jazyk HTML5 zdědil velkou část vlastností a elementů svých předků. Při návrhu této verze bylo totiž dbáno nejen na potřeby budoucích a současných webových stránek, ale velký důraz byl kladen také na zpětnou kompatibilitu. [4]

CSS, celým názvem Cascading Style Sheets, nebo v překladu kaskádové styly se užívají společně s jazykem HTML k tvorbě webových stránek. Zatímco jazyk HTML definuje obsah stránek, tak jazyk CSS určuje spíše podobu obsahu. Stejně jako u jazyka HTML se jedná o textové soubory, jejichž syntaxe je velice jednoduchá a je tedy možné je velice snadno modifikovat.[4]

Bootstrap je volně dostupný HTML, CSS a JS Framework, který je primárně určen k rychlejšímu vyvíjení webů. Obsahuje předdefinované CSS styly pro všechny možné elementy. Není tedy potřeba tvořit vzhled webové stránky úplně od základu, ale stačí pouze přizpůsobovat styly aktuálním potřebám. Kromě předdefinovaných CSS stylů, se ve frameworku nachází také předpřipravené JS scripty.[5]

Doctrine 2 je ORM Framework, celým názvem Object-Relational Mapping Framework pro programovací jazyk PHP. Doctrine zajišťuje mapování relační databáze s pomocí objektů. Mapování probíhá formou PHP tříd, ve kterých je mapování prováděno formou ORM anotací. Proces mapování je obousměrný, to znamená, že je možné vytvořit mapované objekty z již vytvořené databáze, nebo vytvořit databázi z mapovaných objektů. [42]

5.3 Struktura vývojového prostředí



Obrázek 5: Struktura vývojového prostředí, zdroj: vlastní zpracování

Struktura vývojového prostředí je navržena s důrazem kladeným na jednoduchost a přehlednost. Z těchto důvodu se v kořenovém adresáři nacházejí tři podadresáře nazvané *application*, *bin* a *images*.

Prvním popisovaným adresářem je adresář *images*. Obsahem adresáře jsou podadresáře, které se jmenují podle Docker obrazů, které jsou v nich obsaženy. Obrazy jsou vytvořeny prostřednictvím Dockerfile souboru a případných konfiguračních souborů. Podle výše uvedeného obrázku je možné vidět, že adresář obsahuje obrazy s názvem *adminer*, *apache-php*, *composer* a *mariaDB*.

Druhý důležitý adresář se jmenuje *bin*. Jak je již zvykem například z operačních systémů Linux, tento adresář obsahuje příkazy použitelné v rámci vývojového prostředí. Ve skutečnosti tento adresář obsahuje opět pouze podadresáře, které se jmenují podle technologie, pro které jsou příkazy určeny. Příkazy jsou vytvořeny pomocí bash souborů. Důvodem k vytvoření příkazů v tomto formátu bylo volání funkcí jednotlivých kontejnerů. Volání probíhá pomocí Docker příkazů a mnohdy tyto příkazy jsou velice složité, proto ty nejpoužívanější příkazy byly vytvořeny touto formou.

Třetím a posledním adresářem je adresář s názvem *application*. Jak již název napovídá, v tomto adresáři se nacházejí webové aplikace. Na výše uvedeném obrázku je možné vidět, že se v adresáři nacházejí složky s názvem *main* a *work*. Toto rozdělení je pouze ukázkové

a každá z těchto složek obsahuje adresáře s webovými projekty. Tyto složky jsou důležité z důvodu virtuálních hostů, kterými se řídí Apache. Tato problematika je více probrána v kapitolách 5.4.1 a 5.6.1.

Kromě výše zmíněných složek se v kořenovém adresáři nacházejí soubory: gitignore, docker-compose.yml a REAMDE.md. Soubor gitignore se ve zdrojových kódech vývojového prostředí nachází z důvodu verzování verzovacím systémem Git. Do souboru se uvádějí adresáře či soubory, které nepotřebujeme z určitých důvodů verzovat.

Dalším souborem je docker-compose.yml. Tento soubor je jeden z nejdůležitějších v celém vývojovém prostředí. Umožnuje například komunikace mezi izolovanými kontejnery. Soubor bude více popsán v kapitole 5.4.1. Posledním souborem je REAME.md. Tento soubor obsahuje pouze text. Slouží jako soubor informativní a jsou v něm uvedeny informace o vývojovém prostředí.

5.4 Implementace

5.4.1 Tvorba virtualizovaného vývojového prostředí

Je nezbytné virtualizovat jednotlivé technologie, ze kterých je tvořeno celkové vývojové prostředí. V tomto případě se jedná o Apache 2, PHP, Composer, MariaDB a Adminer. Tvorba bude probíhat postupným programováním Dockerfile souborů pro jednotlivé části vývojového prostředí.

První Docker obraz, který slouží jako základ navrhovaného vývojového prostředí se nazývá apache-php. Jak již název napovídá, obraz bude obsahovat technologie Apache a PHP.

```
FROM ubuntu:16.04

# Author
MAINTAINER Dominik Visek <visekdo@gmail.com>

# update installed packages
RUN apt-get update
RUN apt-get -y upgrade

# Install apache,curt,lynx-cur
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install \
        apache2=2.4.18-2ubuntu3.5 curl=7.47.0-1ubuntu2.7 lynx-cur=2.8.9dev8-4ubuntu1
```

Obrázek 6: Dockerfile pro Apache a PHP část 1, zdroj: vlastní zpracování

Na výše uvedeném obrázku je možné vidět první část z Dockerfile souboru pro vytvoření Docker obrazu apache-php. Jako první příkaz, který je na obrázku možné vidět, je příkaz FROM, ve kterém je nadefinováno, že tento Dockerfile vychází z obrazu Ubuntu verze 16.04. Ten je možné nalézt na Docker Hubu. Tento Dockerfile soubor by mohl vycházet i z jiného operačního systému. Vše záleží, na jakém operačním systému chceme, aby byly nainstalovány a spouštěny technologie Apache a PHP. Většinou se volí takový operační systém, který je použit na produkčním serveru. Je zde i možnost vycházet přímo z obrazu Apache. Tento obraz je volně dostupný na DockerHubu pod názvem httpd. Obraz httpd obsahuje již Apache, ale z důvodu osvojení Docker technologie je tento obraz tvořen vlastní implementací této technologie. Po příkazu FROM je nastaven autor obrazu.

Další příkazy, které je možné vidět, jsou příkazy určené pro operační systém Ubuntu a jejich samotné spouštění je implementováno prostřednictvím Docker příkazu RUN. Příkazy,jenž jsou zobrazeny na obrázku, nejprve provedou aktualizace již nainstalovaných balíčků na operačním systému Ubuntu. To vše prostřednitvím balíčkovacího systému APT a poté nainstalují balíčky v následujících verzích: Apache 2 verze 2.4.18, Curl verze 7.47 a Lynx verze 2.8.9. V příkazech se vyskytuje parametr DEBIAN_FRONTEND=nointeractive. Tento parametr znamená, že uživatel nebude žádán o žádné parametry na vyplnění, při samotné instalaci balíčků. Z toho vyplývá, že instalace nebude vůbec přerušována a parametry, které jsou uživatelem běžně při instalaci vyplňovány, budou obsahovat výchozí hodnotu.

```
#packages for add-apt-repository
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python-software-properties
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install software-properties-common
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install python3-software-properties

#set repository
RUN LC_ALL=C.UTF-8 add-apt-repository ppa:ondrej/php
RUN apt-get update

#install php7.2
RUN DEBIAN_FRONTEND=noninteractive apt-get -y install \
        php7.2 php7.2-mysql php7.2-intl php7.2-soap php7.2-bz2 php7.2-curl php7.2-gd php7.2-imap php7.2-xml php7.2-zip

RUN DEBIAN_FRONTEND=noninteractive apt-get -y install \
        php7.2-mbstring php7.2-sqlite
```

Obrázek 7: Dockerfile pro Apache a PHP část 2, zdroj: vlastní zpracování

Po instalaci Apache 2 serveru je také potřeba nainstalovat PHP, které je také součást aktuálního obrazu. Jak již bylo zmíněno Apache a PHP jsou instalovány na operační systém Ubuntu, proto probíhá instalace PHP stejným způsobem, jako by se instalovalo na tento operační systém. Pro potřeby webů je instalováno PHP v aktuálně nejnovější verzi PHP 7.2.

Instalační příkazy jsou opět spouštěny pomocí Docker příkazu RUN. Nejdříve jsou nainstalovány balíčky *python-software-properties*, *software-properties-common* a *python3-software-properties,* které jsou instalovány z důvodu nutnosti použití příkazu *add-apt-repository*. Příkaz *add-apt-repository* slouží k přidání nového repositáře, ze kterého je možné získávat balíčky. V aktuální době se instalace PHP verze 7.2 instaluje z repositáře *ppa:ondrej/php*, který je nutné nejdříve přidat. Po přidání nového repositáře instalace probíhá opět pomocí balíčkovacího systému APT. V Dockerfile souboru se však neinstaluje čisté PHP 7.2, ale instaluje se také mnoho různých rozšíření pro PHP, a to z důvodu případných potřeb vývoje webu, kde jsou tyto rozšíření velice často vyžadovány.

```
# Manually set up the apache environment variables
ENV APACHE_RUN_USER site-data
ENV APACHE_RUN_GROUP site-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid

# Expose apache.
EXPOSE 80

# Update the default apache site with the config we created.
COPY apache-config.conf /etc/apache2/sites-enabled/000-default.conf
COPY php.ini /etc/php/7.2/apache2/php.ini
```

Obrázek 8: Dockerfile pro Apache a PHP část 3, zdroj: vlastní zpracování

Po nainstalování všech potřebných technologií je nutné provést základní konfiguraci. Na výše uvedeném obrázku lze vidět manuální nastavení proměnných pro Apache, kde se pro příklad nastavuje složka pro logovací soubory, což jsou soubory, které zaznamenávají přístupy nebo chybové stavy. Dále je možné vidět nastavení uživatele a skupiny, se kterými je Apache server spouštěn. Po nastavení proměnných je dále nastaven port. Ten je v tomto případě nastaven na 80. Na závěr této části probíhá překopírování konfiguračních souborů pro Apache a PHP. Kopírované soubory jsou umístěny přímo u Dockerfile souboru. Soubor php.ini obsahuje základní konfiguraci PHP, jako jsou třeba povolení PHP rozšíření a nastavení limitů pro paměť. Z důvodu vývoje projektu, které mohou mít rozdílné požadavky pro zprovoznění na PHP rozšíření je povolena většina rozšíření pro PHP.

```
<VirtualHost *:80>
    ServerName localhost.loc:80
    ServerAlias *.loc

    VirtualDocumentRoot "/var/www/site/%-2.1%-2.2%-2.3%-2.4/%-2.5+"
    <Directory /var/www/site>
        AllowOverride All
        Order Allow,Deny
        Allow from All
    </Directory>

   ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Obrázek 9: Apache virtuální hosty, zdroj: vlastní zpracování

Kopírovaný soubor apache-config.conf obsahuje konfiguraci virtuálních hostů. Virtuální hosty se používají pro spuštění více webových stránek s různou doménou na jednom webovém serveru. Vyobrazená konfigurace virtuálních hostů je připravena na přidání libovolného počtu webových stránek s různou doménou. V konfiguračním souboru je nastaveno, aby se cesta ke zdrojovým souborům webových stránek řídila doménou, jak je možné vidět v parametru *VirtualDocumentRoot,* kde je definována cesta k webům. Cesta je pevně definována do adresáře *site*, odkud je dále cesta definována za pomocí domény webu. První čtyři znaky z domény definují podadresář, ve kterém je daný projekt a zbytek názvu z domény označuje složku projektu. Tento způsob virtuálních hostů je navržen z důvodu případného rozdělení projektů vývojáře například na vlastní projekty a projekty do práce. Patřičné příklady jsou zobrazeny v kapitole 6.4.1. Dále jsou v souboru nakonfigurována umístění pro logovací soubory a další potřebná nastavení virtuálních hostů, zejména kvůli *.htaccess* souborům. Tyto *.htaccess* soubory jsou konfigurační soubory pro http server, které lze například nastavit přesměrovaní do jiných adresářů a mnoho dalšího.

```
# Enable apache mods.
RUN a2enmod php7.2
RUN a2enmod rewrite
RUN a2enmod vhost_alias

RUN service apache2 restart

# By default start up apache in the foreground, override with /bin/bash for interative.
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

Obrázek 10: Dockerfile pro Apache a PHP část 4, zdroj: vlastní zpracování

V poslední části Dockerfile souboru se nacházejí povolení pro Apache. Jedná se o povolení jazyka PHP 7.2, modulu rewrite a vhost_alias. Modul rewrite slouží zejména pro práci s url adresami a modul vhost_alias umožnuje dynamická nastavení virtuálních hostů prostřednictvím *VirtualDocumentRoot*. Po povolení potřebných modulů je potřeba restartovat server z důvodu aplikování změn. Po tomto kroku následuje poslední příkaz z aktuálního Dockerfile souboru, který spouští Apache server v popředí.

Pro tvorbu dalších potřebných obrazů byly využity již vytvořené obrazy, které jsou volně dostupné v repositáři Docker Hub.

```
FROM mariadb:10.3.5

# Author
MAINTAINER Dominik Visek <visekdo@gmail.com>
```

Obrázek 11: Dockerfile mariaDB, zdroj: vlastní zpracování

Každý další vytvořený Dockerfile soubor vypadá jako na Obrázku 11 Dockerfile pro MariaDB databázi. Dockerfile čerpá z daného obrazu určité verze, v tomto případě je to obraz Mariadb verze 10.3.5. Následně se také vyskytuje příkaz pro definování autora aktuálního Dockerfile souboru. Pouze Dockerfile pro composer je odlišný z důvodu volání příkazů pro Composer. Proto v obrazu pro Composer je přidán bash script nekonečné smyčky, aby bylo možné příkazy využívat.[30]

Po vytvoření jednotlivých Docker obrazů je potřeba nastavit komunikaci a sdílení dat mezi jednotlivými kontejnery. Webové stránky jsou umístěny v jednom kontejneru a musí být schopné se mimo jiné připojit k databázi, která je umístěna v jiném izolovaném kontejneru. Je tedy potřeba navrhnout soubor se stejnojmenným názvem, tedy „docker-compose.yml“. Pro ukázku jsou zvoleny dvě části kódu ze jmenovaného souboru.

```yaml
db:
  build: ./images/mariaDB
  image: bc_visek_db
  container_name: bc_visek_db
  restart: always
  ports:
    - 3306:3306
  volumes:
    - ./images/mariaDB/data:/docker-entrypoint-initdb.d
    - persistent:/var/lib/mysql
  environment:
    MYSQL_ALLOW_EMPTY_PASSWORD: 1
```

Obrázek 12: Ukázka docker-compose.yml databáze, zdroj: vlastní zpracování

První ukázka zobrazuje kód pro nastavení služby *db*. Z výše uvedeného kódu je možné vyčíst, že služba *db* provádí vytvoření obrazu z Dockerfile souboru, ke kterému je definována cesta v parametru *build*. Při sestavování obrazu a vytváření kontejnerů se výchozí název jednotlivých obrazů a kontejnerů nastavuje dle adresáře, ve kterém se docker-compose.yml nachází a dále dle názvu služby. Vzhledem k tomu, že pro ovládání základních nástrojů jsou vytvořeny příkazy formou bash souborů, v případě přejmenování složky projektu by příkazy v bash souborech nefungovaly. Z tohoto důvodu je vyplněn parametr image a container_name, který v tomto případě definuje název obrazu a jeho kontejneru. Dále je zde nastaven port, volumes a environments. Jinými slovy, port na kterém databáze naslouchá; připojení svazku s inicializačními sql soubory a nastavení proměnných hodnot pro přístup k databázi, kde je nastaveno povolení se připojovat k databázi pod uživatelem root bez hesla, což je pro lokální vývoj plně dostačující.

```yaml
server:
  build: ./images/apache-php
  image: bc_visek_server
  container_name: bc_visek_server
  working_dir: /server/application
  restart: always
  volumes:
    - ./:/server
    - ./application:/var/www/site
  links:
    - db
    - composer
  ports:
    - 80:80
```

Obrázek 13: Ukázka docker-compose.yml server, zdroj: vlastní zpracování

Druhý příklad je velice podobný prvnímu. Jedná se o nastavení služby *server,* která vychází z Dockerfile souboru, který obsahuje Apache a PHP. V konfiguraci služby server zde oproti službě *db* přibývá atribut *working_dir*. Parametr říká, že všechny prováděné příkazy ve vytvořeném kontejneru budou mít nastavený výchozí adresář, ve kterém budou příkazy prováděny. V tomto případě je jako výchozí adresář nastaven adresář obsahující webové aplikace. Dále je zde připojení svazků za pomocí *volumes*. Svazky s projekty jsou připojeny k adresáři, ve kterém http server hledá webové aplikace. Pokud by byl adresář pouze překopírován do té složky, změny provedené na webových stránkách v rámci kódu by se na webu projevily až po znovu překopírování stejného projektu, zatímco takto propojené svazky dovolují změny v reálném čase. Posledním atribut, který je zde nový oproti službě *db*, je atribut *links*. Atribut *links* nám říká, které služby je možné volat z aktuálního kontejneru. V tomto případě se jedná o db a composer. Další konfigurace služeb zde nebude vyobrazena, je totiž velice podobná již ukázaným konfiguracím. Pro případné další prozkoumání je možné soubor nalézt v kořenovém adresáři virtualizovaného vývojového prostředí.

5.4.2 Tvorba Symfony projektu

Projekt bude obsahovat jednoduchý seznam úkolů. Jako první bod při tvorbě projektu za použití Symfony frameworku je provedení instalace Symfony frameworku. Ve skutečnosti se nejedná o instalaci, ale pouze o stažení zdrojových souborů frameworku. K instalaci frameworku se využívá nástroj s názvem Composer. K instalaci projektu stačí spustit bash soubor, který framework do patřičného adresáře nainstaluje. Spuštění bashsouboru je možné vidět v následujícím příkazu. Příkaz je spouštěn z kořenového adresáře projektu.

```
bash bin/composer/composer-create-project.sh symfony/skeleton project work
```

Po provedení instalace, již není potřeba nastavovat virtuální hosty, jelikož je vše předpřipraveno. Je však potřeba přidat řádek do souboru hosts, který vypadá následovně.

```
127.0.0.1       workproject.loc www.workproject.loc
```

Dále je do projektu přidán *htaccess* soubor, který přesměrovává z kořenového adresáře projektu do adresáře, ten obsahuje soubor index.php. Důvodem řešení tohoto problému pomocí *htaccess* souboru je, že v různých frameworcích je tento soubor v různých složkách.

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /

    RewriteRule ^(.*)$ %{ENV:BASE}web/$1 [QSA,L]
</IfModule>
```

Obrázek 14: .htaccess pro Symfony projekt, zdroj: vlastní zpracování

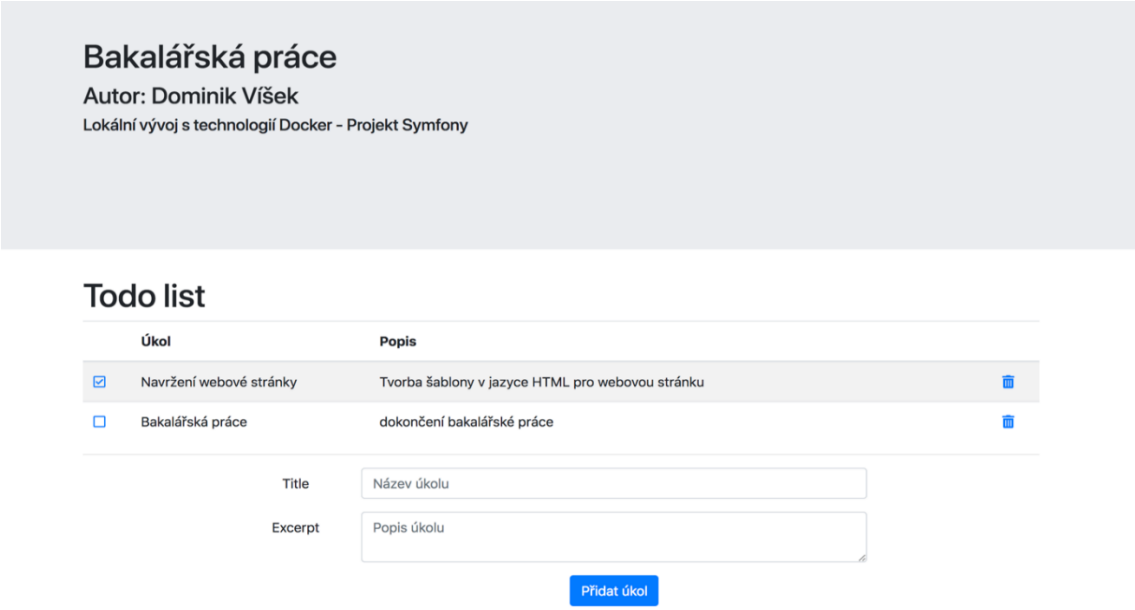
Pro připojení k databázi je využit framework *Doctrine 2*. Bylo tedy potřeba navrhnout entity, které představují databázové tabulky. V rámci projektu je vytvořena pouze jedna tabulka, ve které budou obsaženy jednotlivé úkoly ze seznamu úkolů.

```
/**
 * TodoList
 *
 * @ORM\Table(name="todo_item")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\TodoItemRepository")
 * @ORM\HasLifecycleCallbacks
 */
class TodoItem extends BaseEntity
{

    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @GeneratedValue
     */
    private $id;
```

Obrázek 15: Ukázka části Doctrine entity, zdroj: vlastní zpracování

Po navržení entity a vytvoření tabulky v databázi je provedena implementace, která spočívá v tvorbě vzhledu webové stránky a vytvoření metod a formulářů pro správnou funkčnost. Pro vytvoření vzhledu byl použit *Twitter bootstrap* verze 4 a *font-awesome*, který obsahuje různá písma a responzivní ikony. Vykreslování stránky je implementováno v *DefaultControlleru*, ve kterém jsou předány všechny pomocné služby pro běh aplikace. HTML značky pro vykreslování stránky jsou definovány v twig šablonách. Výsledná webová stránka je ukázána na následujícím obrázku.



Obrázek 16: Ukázka vytvořené webové stránky, zdroj: vlastní zpracování

Vzhled webové stránky je navržen s důrazem na jednoduchost, přehlednost a responzivitu. Stránka je tvořena hlavičkou, kde jsou uvedeny základní informace, jako jsou: autor nebo použitý framework na kterém je webová stránka vybudována. Tělo následně tvoří seznam úkolů s formulářem pro přidávání úkolů do seznamu.

V rámci Symfony projektu byly dále implementovány *PHPUnit* testy. *PHPUnit* testy obsahují nástroje pro otestování aplikací založených zejména na PHP frameworcích. Pokud je programována webová aplikace, vždy jsou spolu s ní programovány i testy pro jednotlivé metody. Motivací pro psaní testů je zejména funkčnost aplikace. Pokud se modifikují kódy, které byly napsány v minulosti, tak se dost často objevují chyby způsobené právě zmíněnou modifikací kódu. Pokud jsou testy napsány správně, kód by neprošel testy, dokud by v kódu nebyly odladěny veškeré nedostatky. Spuštění testů je možné za pomocí bash souboru.[29]

```php
public function testIndex()
{
    $client = static::createClient();

    $crawler = $client->request( method: 'GET', uri: '/');

    $this->assertEquals( expected: 200, $client->getResponse()->getStatusCode());
    $this->assertContains( needle: 'Todo list  | Symfony project', $crawler->filter( selector: 'title')->text());
}
```

Obrázek 17: Ukázka testovací metody za použití phpunit, zdroj: vlastní zpracování

5.4.3 Tvorba Nette projektu

Projekt vytvořený za pomocí Nette frameworku bude obsahovat úplně to stejné co projekt vytvořený za pomocí Symfony frameworku. Instalace frameworku je provedena následujícím příkazem.

```
bash bin/composer/composer-create-project.sh nette/web-project project main
```

Také zde musí být přidán .htaccess soubor, který slouží k přesměrování do složky, ve které se nachází soubor index.php.

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /

    RewriteRule ^(.*)$ %{ENV:BASE}www/$1 [QSA,L]
</IfModule>
```

Obrázek 18: .htaccess pro Nette projekt, zdroj: vlastní zpracování

Po vytvoření htaccess souboru je nutné přidat řádek do souboru hosts, který vypadá následovně.

```
127.0.0.1       mainproject.loc www.mainproject.loc
```

K připojení k databázi je použit Framework Doctrine 2. Vykreslovní stránky probíhá v HomepagePresenteru, ve kterém jsou předány všechny pomocné služby pro běh aplikace. HTML značky pro vykreslování stránky jsou definovány v latte šablonách. Výsledný vzhled webové stránky je stejný jako u Symfony projektu, pouze se liší text s použitým frameworkem. Pro doplnění informací, PHPUnit testy jsou implementovány pouze v Symfony frameworku.

5.5 Testování

Testování virtualizovaného vývojového prostředí probíhá formou webových stránek. Vytvořené webové stránky využívají kontejnery *bc_visek_server* a *bc_visek_db*, které musí spolupracovat, aby weby správně fungovaly. Dále je možné využít kontejner *bc_visek_composer*, který může fungovat i samostatně. Kontejner se ve virtuálním vývojovém prostředí vyskytuje z důvodu instalací webových Frameworků a potřebných PHP knihoven. Poslední kontejner je kontejner pro Adminer. Musí být možné se připojit z bc_visek_adminer kontejneru do databáze, která je v kontejneru jiném.

5.6 Příručka

5.6.1 Zprovoznění vývojového prostředí

Pro zprovoznění vývojového prostředí musí být nainstalován Docker a Docker Compose. Tvorba prostředí probíhala v technologii Docker verze18.03.0-ce a Docker Compose verze 1.20.1. Pokud jsou tyto předpoklady splněny, pak spuštění samotného projektu je možné provést dvěma způsoby. Pro spuštění prostředí je nejdříve nutné přistoupit do kořenového adresáře vývojového prostředí pomocí příkazové řádky příkazem *cd*. Po přístupu do adresáře je možné projekt spustit dvěma způsoby. Pomocí příkazu technologie Docker Compose, nebo pomocí předpřipraveného bash skriptu spouštěného z kořenového adresáře vývojového prostředí. Příkazy vypadají následovně.

```
docker-compose up                          #spuštění za pomocí Docker Compose

bash bin/docker/build-run.sh               #spuštění za pomocí bash souboru
```

Po spuštění projektu budou weby zatím nefunkční. Pro zprovoznění webů je nezbytné nastavit hosty pro vytvořené weby. Soubor s hosty se nachází na těchto adresách.

```
MacOS/Linux:     /etc/hosts

Windows:         /Windows/System32/drivers/etc/hosts
```

Do hosts souboru je potřeba přidat následující řádky.

```
127.0.0.1        mainproject.loc www.mainproject.loc

127.0.0.1        workproject.loc www.workproject.loc
```

Cesta k webu se definuje pomocí virtuálních hostů. V rámci vývojového prostředí jsou virtuální hosty nastavené tak, aby cesta k samotným projektům byla dynamicky určena dle domény. Pokud tedy doména odpovídá například doméně *mainproject*, první čtyři znaky, tedy *main* je adresář, ve kterém se nachází daný projekt a zbytek z domény je název samotného projektu. Z uvedené domény lze tedy vyčíst, že adresář projektu se nachází na následující adrese operačního systému Ubuntu.

```
/var/www/site/main/project
```

Cesta /var/www/site je nastavena staticky a zbytek cesty je určen z domény.

5.6.2 Bash soubory

Bash soubory byly vytvořeny za účelem jednoduššího a intuitivnějšího ovládání celého vývojového prostředí. Příkazy jednotlivých technologií jsou volány za pomocí Docker technologie, protože instalované technologie se nacházejí v jednotlivých kontejnerech, a proto jsou příkazy zdlouhavé a občas i složité. Bash soubory jsou umístěny v kořenovém adresáři ve složce bin, kde jsou další adresáře s názvem technologií, pro které jsou příkazy určeny. Pro spuštění bash souborů na operačním systému Windows je nutné si doinstalovat Linux Bash. Použití souborů je velice jednoduché a vypadá následovně.

```
bash <cesta_k_souboru> <parameter 1> <parameter 2> .. #záleží na vybraném souboru
```

Přehled parametrů a souborů je možné vidět v následujících tabulkách: **Tabulka 4**, **Tabulka 5**, **Tabulka 6**, které se nacházejí níže. **Tabulka 4** obsahuje přehled souborů, které slouží k ovládání nástroje composer. **Tabulka 5** obsahuje soubory pro ovládání phpunit testů a **Tabulka 6** obsahuje přehled souborů pro ovládání virtualizovaného vývojového prostředí za pomocí Dockeru. Jako výchozí adresář pro všechny cesty k projektům je použit adresář, ke kterému vede následující cesta /var/www/site.

Název souboru a parametry	Účel souboru
composer-create-project.sh <framework> <název_projektu> <umístění>	Soubor slouží zejména k instalaci frameworků. První parametr odkazuje na Framework a je možné ho vyčíst z dokumentací frameworků. Druhý parametr je název projektu, který bude vytvořen a poslední parametr je umístění projektu.
composer-init.sh <adresář_projektu>	Soubor sloužící k inicializaci Composeru v projektu. Jediný obsažený parametr slouží k nalezení projektu, který má být inicializován.
composer-install.sh <adresář_projektu>	Instalace nenainstalovaných závislostí dle souboru composer.json.
composer-remove.sh <balíček> <adresář_projektu>	Odstranění nainstalovaného balíčku z určitého projektu.
composer-require.sh <balíček> <adresář_projektu>	Instalace nového balíčku do určitého projektu.
composer-search.sh <hledáno> <adresář projektu>	Vyhledání balíčku k nainstalování.
composer-update.sh <adresář_projektu>	Aktualizace nainstalovaných balíčků a následná instalace nových verzí balíčků.

Tabulka 4: Přehled Composer bash souborů

Název souboru a parametry	Účel souboru
phpunit-run-tests.sh <adresář_projektu> <cesta_ke_specifickým_testům>	Slouží ke spouštění phpunit testů. První parametr slouží k přístupu do projektu, ve kterém mají být testy spouštěny. Druhý parametr slouží ke spuštění specifického adresáře s testy nebo přímo ke spuštění jednoho testu. Pokud není druhý parametr vyplněn, jsou spouštěny všechny testy z daného projektu.

Tabulka 5: Přehled phpunit bash souborů

Název souboru a parametry	Účel souboru
build-run.sh	Spuštění Docker projektu se znovusestavením.
remove-all-containers.sh	Odstranění všech kontejnerů.
remove-all-images.sh	Odstranění všech obrazů
stop.sh	Ukončení běžících kontejnerů

Tabulka 6: Přehled Docker bash souborů

ZÁVĚR

Cílem teoretické části této práce bylo uvedení čtenáře do problematiky virtualizace a následné představení technologie Docker. V průběhu tvorby teoretické části práce bylo potřeba představit a popsat i technologie Docker compose, která byla velice důležité pro tvorbu praktické části bakalářské práce.

Cíl teoretické části byl úspěšně splněn. Čtenář byl uveden do problematiky virtualizace v kapitole 1 a seznámen s technologií Docker a Docker Compose v kapitole 2. V práci byly také představeny některé alternativy k technologii docker. Tyto technologie je možné nalézt v kapitole 3.

Cílem praktické části bakalářské práce bylo vytvoření virtualizovaného vývojového prostředí pro lokální vývoj za použití technologie Docker. Důležité bylo navrhnout a implementovat prostředí tak, aby bylo co nejvíce intuitivní. Z tohoto byl kladen velký důraz při návrhu prostředí na snadnou ovladatelnost, aby samotný vývoj webových aplikací za pomocí jazyka PHP a případných použitých frameworků byl co nejvíce uživatelsky přívětivý.

Návrh virtualizovaného vývojového prostředí i následná implementace se vydařila a tato problematika je důkladně popsána v kapitole 5. Vytvořené lokální vývojové prostředí bylo testováno na operačních systémech Ubuntu, Mac OS a Windows a ve všech případech virtualizované prostředí plně fungovalo. Dále byla sepsána uživatelská příručka pro dané virtualizované prostředí, která se nachází v kapitole 5.6. V uživatelské příručce je popsáno jak ovládání prostředí, tak i práce s jednotlivými webovými projekty.

Jedním z hlavních cílů této práce bylo osvojení technologie Docker. Autor této bakalářské práce před samotnou tvorbou neměl žádné zkušenosti s Docker technologií. Praktická část práce přinesla autorovi mnoho zajímavých a nových znalostí a zkušeností. Vytvoření praktické části práce bylo časově náročnější, než byl původní odhad, ovšem výsledek a cíl byl v závěru splněn.

Další směr bakalářské práce lze spatřovat v rozšířených možnostech technologie Docker, která se stále v současné době vyvíjí.

POUŽITÁ LITERATURA

[1] About Docker CE. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/install/

[2] Best practices for writing Dockerfiles. *Docker docs* [online]. 2018-04-29 [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

[3] BRODKIN, Jon. With long history of virtualization behind it, IBM looks to the future. *Network World* [online]. Network World, 2009-04-30 [cit. 2018-04-30]. Dostupné z: https://www.networkworld.com/article/2254433/virtualization/with-long-history-of-virtualization-behind-it--ibm-looks-to-the-future.html

[4] CASTRO, Elizabeth a Bruce HYSLOP. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. Brno: Computer Press, 2012. ISBN isbn-978-80-251-3733-8.

[5] ČÁPKA, David. 1. díl - Úvod do CSS frameworku Bootstrap. *Itnetwork.cz* [online]. itnetwork.cz [cit. 2018-04-30]. Dostupné z: https://www.itnetwork.cz/html-css/bootstrap/uvod-do-css-frameworku-bootstrap

[6] Čím je Adminer lepší než phpMyAdmin? *Adminer* [online]. Překlad: VRÁNA, Jakub. 2017 [cit. 2017-11-06]. Dostupné z: https://www.adminer.org/cs/phpmyadmin

[7] Docker. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/engine/reference/commandline/docker/

[8] Docker Compose. *Docker docs* [online]. 2018-04-29 [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/compose/https://docs.docker.com/compose/

[9] .Docker overview. *Docker docs* [online]. 2018-04-29 [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/engine/docker-overview/

[10] Docker Registry. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/registry/

[11] Docker RUN vs CMD vs ENTRYPOINT. *Zdroják* [online]. 2016-05-02 [cit. 2018-04-30]. Dostupné z: http://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/

[12] Dockerfile reference. *Docker docs* [online]. 2018-04-29 [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/engine/reference/builder/

[13] Docker security. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/engine/security/security/

[14] Dockerd. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/engine/reference/commandline/dockerd/

[15] Docker: Up and Running, 2015, O'Reilly Media, Incorporated, ISBN: 978-14-9191-757-2 Mouat Addrian

[16] Documentation. *Golang* [online]. Google [cit. 2018-04-30]. Dostupné z: https://golang.org/doc/

[17] FI MU: Virtualizace a kontejnerizace. *Fakulta informatiky Masarykovy univerzity: Virtualizace a kontejnerizace* [online]. Brno: Fakulta informatiky Masarykovy univerzity, 1995 [cit. 2017-12-12]. Dostupné z: http://www.fi.muni.cz/~kas/pv090/referaty/2014-podzim/virt.html

[18] FI MU: Virtualizace. *Fakulta informatiky Masarykovy univerzity:*[online]. Brno: Fakulta informatiky Masarykovy univerzity, 1995 [cit. 2018-04-30]. Dostupné z: https://www.fi.muni.cz/~kas/pv090/referaty/2016-podzim/virt.html

[19] Get Started, Part 1: Orientation and setup. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/get-started

[20] GOTO, Daichi. The new unionfs implementation for FreeBSD and status of merging. *Freebsd* [online]. 2006-01-04 [cit. 2018-04-30]. Dostupné z: https://people.freebsd.org/~daichi/unionfs/

[21] History of Virtualization. *I Dont Know, Read The Manual* [online]. IDKRTM, 2018-01-25 [cit. 2018-04-30]. Dostupné z: https://www.idkrtm.com/history-of-virtualization/

[22] Get Docker CE for Ubuntu. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/install/linux/docker-ce/ubuntu/

[23] Install Docker for Mac. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/docker-for-mac/install/

[24] Install Docker for Windows. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/docker-for-windows/install/

[25] Introduction to User Namespaces in Docker Engine. *Docker success center* [online]. [cit. 2018-04-30]. Dostupné z: https://success.docker.com/article/introduction-to-user-namespaces-in-docker-engine

[26] Introduction to Vagrant. *Vagrant* [online]. [cit. 2018-04-30]. Dostupné z: https://www.vagrantup.com/intro/index.html

[27] Jails – High value but shitty Virtualization¶. *Freebsd* [online]. [cit. 2018-04-30]. Dostupné z: http://phk.freebsd.dk/sagas/jails.html

[28] Luděk Matyska. Techniky virtualizace počítačů (2). Zpravodaj ÚVT MU. ISSN 1212-0901, 2007, roč. XVII, č. 3, s. 9-12.

[29] Machek Zdenek, PHPUnit Essentials, 2014, Packt Publishing Ltd., ISBN: 978-17-8328-344-6 Matthias Karl, Kane Sean P.,

[30] Official repository mariaDB. *Docker Hub* [online]. Docker [cit. 2018-04-30]. Dostupné z: https://hub.docker.com/_/mariadb/

[31] Overview of docker-compose CLI. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/compose/reference/overview/

[32] PETROFČÍK, Matúš. Composer. *Itnetwork.cz* [online]. itnetwork.cz [cit. 2018-04-30]. Dostupné z: https://www.itnetwork.cz/php/ostatni/composer

[33] PIKE, Rob. Go at Google: Language Design in the Service of Software Engineering. *Golang* [online]. Google [cit. 2018-04-30]. Dostupné z: https://talks.golang.org/2012/splash.article#TOC_1.

[34] POMAZAL, Jiří. Virtualizace v kostce. *Systemonline* [online]. [cit. 2018-04-30]. Dostupné z: https://www.systemonline.cz/clanky/virtualizace-v-kostce.htm

[35] POŠMURA, Vlastimil. *Apache: příručka správce WWW servru*. Praha: Computer Press, 2002. Všechny cesty k informacím. ISBN isbn80-7226-696-9.

[36] PREIMESBERGER, Chris. *IT Science Case Study: Consolidating to a Single Virtual Platform* [online]. eweek, 2017-12-29 [cit. 2018-05-08]. Dostupné z: http://www.eweek.com/innovation/it-science-case-study-consolidating-to-a-single-virtual-platform

[37] Pricing. *Docker* [online]. Docker, 2017-02-27 [cit. 2018-05-08]. Dostupné z: https://www.docker.com/pricing

[38] Runtime metrics. *Docker docs* [online]. [cit. 2018-04-30]. Dostupné z: https://docs.docker.com/config/containers/runmetrics/

[39] ROUSE, Margaret. What is Docker Engine?. *TechTarget* [online]. 2017-10-17 [cit. 2018-04-30]. Dostupné z: https://searchitoperations.techtarget.com/definition/Docker-Engine

[40] SVAČINKA, Martin. Úvod do Vagrantu. *Zdroják* [online]. Praha: Zdroják, 2015-08-15 [cit. 2018-04-30]. Dostupné z: https://www.zdrojak.cz/clanky/uvod-do-vagrantu/

[41] ŠTRAUCH, Adam. Databáze MariaDB válcuje MySQL. *Root.cz* [online]. 21. 12. 2012 [cit. 2018-05-01]. ISSN 1212-8309. Dostupné z: https://www.root.cz/clanky/databaze-mariadb-valcuje-mysql

[42] TICHÝ, Jan. Doctrine 2: úvod do systému. *Zdroják* [online]. 2010-07-21 [cit. 2018-04-30]. Dostupné z: https://www.zdrojak.cz/clanky/doctrine-2-uvod-do-systemu/

[43] Using Docker: Developing and Deploying Software with Containers, 2015, O'Reilly Media, Inc. ISBN: 978.14.-9191-591-2 Krochmalski Jaroslaw

[44] Vagrant vs. Docker - Vagrant by HashiCorp. *Vagrant* [online]. [cit. 2018-04-30]. Dostupné z: https://www.vagrantup.com/intro/vs/docker.html

[45] Washraf. Union File System. *Gitbook* [online]. Lyon: GitBook [cit. 2018-04-30]. Dostupné z: https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%203/union_file_system.html

[46] WELLING, Luke. *PHP a MySQL: rozvoj webových aplikací*. 3. vyd. Praha: SoftPress, 2005. ISBN isbn-80-86497-83-6.

[47] Windocks: a Modern, Open, Data Delivery Platform. In: *Windocks* [online]. [cit. 2018-04-30]. Dostupné z: https://www.windocks.com/files/Windocks-Docker-Sql-Server-Containers-Technical-Backgrounder.pdf

[48] 23.YEGULALP, Serdar. WinDocks does what Docker and Microsoft can't do. *InfoWorld* [online]. 2017-04-04 [cit. 2018-04-30]. Dostupné z: https://www.infoworld.com/article/3051588/application-virtualization/windocks-does-what-docker-and-microsoft-cant-do.html

[49] ŽOLTÁ, Lucie. *Disciplína sběr a analýza požadavků* [online]. [cit. 2018-05-08]. Dostupné z: http://lucie.zolta.cz/index.php/softwarove-inzenyrstvi/150-disciplina-sber-a-analyza-pozadavku

PŘÍLOHY

PŘÍLOHA A – DIAGRAM VÝVOJE SOFTWARU

PŘÍLOHA B – PŘILOŽENÉ CD

Přiložené CD obsahuje:

- kód virtualizovaného vývojového prostředí s webovými projekty,
- PDF verzi bakalářské práce.