Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Kontejnerová virtualizace s Docker a Docker Swarm na Raspberry Pi

Jakub Jan

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jakub Jan

Informatika

Název práce

Kontejnerová virtualizace s Docker a Docker Swarm na Raspberry Pi

Název anglicky

Container virtualization with Docker and Docker Swarm on Raspberry Pi

---

Cíle práce

Hlavním cílem práce je nasazení kontejnerové virtualizace ve spojení s platformami Docker a Docker Swarm na cluster sestavený z mikropočítačů Raspberry Pi.

Metodika

Metodika řešení práce je založena na studiu a analýze dostupných odborných informačních zdrojů. Základem pro vypracovaní práce bude fyzické vytvoření clusteru z mikropočítačů Raspberry Pi, na které bude nainstalována platforma Docker pro kontejnerovou virtualizaci. Tyto mikropočítače budou poté propojeny pomocí Docker Swarm do virtualizačního clusteru. Závěrečná demonstrace možností celého řešení bude web hosting s vysokou dostupností a load balancingem napříč clusterem.

Doporučený rozsah práce

30-40 stran

Klíčová slova

kontejnerová virtualizace, container virtualization, Docker, Raspberry Pi

---

Doporučené zdroje informací

MATTHIAS, Karl a Sean P. KANE. Docker: Up & Running: Shipping Reliable Containers in Production. O'Reilly Media. ISBN 978-1491917572.

SOPPELSA, Fabrizio a Chanwit KAEWKASI. Native Docker Clustering with Swarm. Packt Publishing 2016. ISBN 9781786469755.

TURNBULL, James. The Docker Book. 2014. ISBN 978-0-9888202-0-3.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 04. 02. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Kontejnerová virtualizace s Docker a Docker Swarm na Raspberry Pi" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2018 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Marek Pícka, Ph.D. za jeho čas a odborné vedení při zpracování této práce. Také bych rád poděkoval své rodině a přátelům za morální podporu.

Kontejnerová virtualizace s Docker a Docker Swarm na Raspberry Pi

Abstrakt

Tato bakalářská práce věnuje problematice kontejnerové virtualizace a jejím hlavním cílem je nasazení kontejnerové virtualizace na cluster sestavený z mikropočítačů Raspberry Pi. Virtualizace na jednotlivých nodech bude zajištěna pomocí platformy Docker. Docker Swarm bude poté sloužit k přidání možnosti sloučit jednotlivé nody do clusteru se všemi jeho výhodami.

Klíčová slova: Kontejnerová virtualizace, Docker, Raspberry Pi, Docker Swarm, Linux, Virtualizace, Cluster, Orchestrace

Container virtualization with Docker and Docker Swarm on Raspberry Pi

Abstract

This bachelor thesis deals with the issue of container virtualization and its main goal is to deploy container virtualization to a cluster made of Raspberry Pi microcomputers. Virtualization on individual nodes will be achived via the Docker platform. The Docker Swarm will then be used to assemble nodes to cluster with all its benefits.

Keywords: Container virtualization, Docker, Raspberry Pi, Docker Swarm, Linux, Virtualization, Cluster, Orchestration

Obsah

1 Úvod 11

2 Cíl práce a metodika 12

3 Virtualizace 13

3.1 Host 13

3.2 Hypervisor 14

3.2.1 Nativní hypervizor 14

3.2.2 Hostovaný hypervizor 14

3.3 Virtual machine (VM) 15

3.3.1 Snapshot 15

3.4 Využití virtualizace 16

3.4.1 Desktopové využití 16

3.4.2 Serverové využit 16

3.5 Výhody virtualizace 17

3.6 Historie virtualizace 18

3.7 Příklad podnikového nasazení 19

4 Kontejnerová virtualizace 21

4.1 Výhody kontejnerové virtualizace 21

4.2 Historie kontejnerové virtualizace 22

4.3 “Mazlíček nebo Stádo” 22

5 Docker 24

5.1 Komponenty Dockeru 24

5.2 Výhody Dockeru 25

5.3 Životní cyklus kontejneru v Dockeru 26

5.4 Docker Swarm 27

6 Raspberry Pi cluster 28

6.1 Raspberry Pi 28
6.2 Ostatní komponenty 30
6.2.1 SD Karty 30
6.2.2 Switch/Router 30
6.2.3 Síťové karty 30
6.2.4 USB hub 31
6.2.5 Cenová kalkulace 32
6.3 Sestavení 32
6.4 Instalace a nastavené operačního systému 34
6.4.1 Instalace 34
6.4.2 Po instalační kroky 34
7 Instalace Docker 36
7.1 Nastavení Docker Swarm 36
8 Použité kontejnery 38
8.1 Compose File 39
9 Zhodnocení přínosu využití platformy Docker 41
10 Závěr 42

Seznam obrázků

Obrázek 1: Zjednodušené aplikační schéma [z1] 11

Obrázek 2: Znázornění rozdílu tradičního a virtualizačního přístupu [z1] 13

Obrázek 3: Znázornění rozdílu plné a kontejnerové virtualizace [z1] 21

Obrázek 4: Logo aplikace Docker [z2] 24

Obrázek 5: Fotka Raspberry Pi Zero s instalovaným pasivním chladičem [z1] 29

Obrázek 6: Fotka finálního clusteru bez napájecích zdrojů [z1] 33

Seznam tabulek

Tabulka 1: Přehled vybraných hypervizorů 14

Tabulka 2: Seznam a vlastnosti serverů aplikací před konsolidací 20

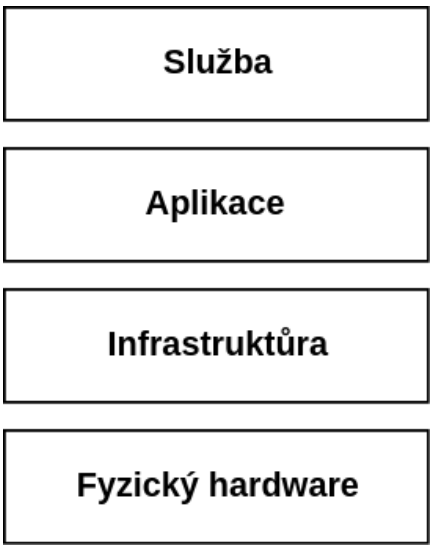
Tabulka 3: Porovnání aktuálně prodávaných modelů Raspberry Pi 28

Tabulka 4: Výsledná cenová kalkulace 32

1 Úvod

Mezi nejzajímavější nové technologie posledních let bezesporu patří kontejnerová virtualizace. Tato technologie dokázala zaujmout velkou část open source komunity, stejně jako velké hráče na poli IT byznysu jako jsou třeba Google, Amazon, Vmware nebo Intel. Hlavním tahákem kontejnerové virtualizace je že plynule navazuje na myšlenku "as a Service" neboli česky "jako služba". Tato myšlenka je založena na tom, že veškeré složky IT byznysu je možné poskytovat jako službu, ať je to od nejspodnějšího vrstvy aplikačního schéma, poskytování hardware jako "Metal as a Service", zde patří například klasický pronájem dedikovaného serveru, nebo nejvyšší vrstvu kde je celá služba jako služba, kde za nejlepší příklad považuji Google Dokumenty od Google.

Kontejnerová virtualizace na platformě Docker usnadňuje především poskytovaní středních a vyšších vrstev aplikačního schéma jako infrastruktury, aplikace a samotnou službu. Práce s infrastrukturou je usnadněna, neboť veškerá infrastruktura nad clusterem je Docker a Docker Swarm striktně definována jako kód. Poskytování aplikací a služeb, jak služeb je poté usnadněno už v samotné filosofii těchto dvou platforem a to je, že každá aplikace by měla být škálovatelná "do boku", neboli přidáním replik aplikace.



Obrázek 1: Zjednodušené aplikační schéma [z1]

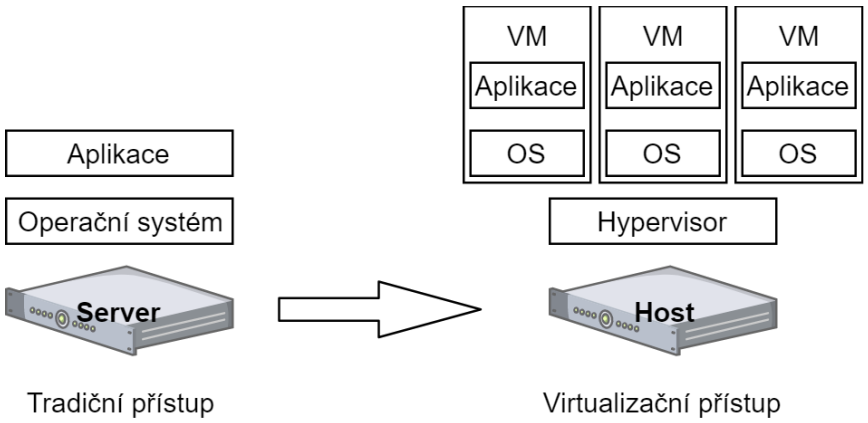
2 Cíl práce a metodika

První cíl bakalářské práce je popis kontejnerové virtualizace a její porovnání oproti standardní virtualizaci. Hlavním cílem práce je poté nasazení kontejnerové virtualizace na cluster sestavený z mikropočítačů Raspberry Pi.

Práce se skládá ze dvou částí, přesněji teoretické a praktické. V teoretické části bude pokryt převážně výše uvedený první cíl práce za dostupných odborných informačních zdrojů a publikací. Praktická část se bude postupně zabývat návrhem clusteru z mikropočítačů Raspberry Pi, jeho fyzickou realizací, přípravou operačního systému, instalací platformy Docker, konfigurací Docker Swarm pro vytvoření clusteru, úpravou standardních aplikací webového serveru pro běh v kontejneru a jejich finálním nasazením. Tato praktická část by měla poskytnout jednoduchý a přenosný nástroj, na kterém je možné prezentovat vlastnosti kontejnerové virtualizace a clusteru širší odborné veřejnosti.

3 Virtualizace

Virtualizace je technologie umožňující vytvoření a současný běh několika oddělených a na sobě nezávislých softwarových prostředí na jednom hardware. Jednoduše řečeno, ze softwarového pohledu se jeden systém rozdělí na několik [c1]. Každé ze softwarových prostředí má přidělené vlastní hardware zdroje jako jsou, CPU, RAM, síťová nebo grafická karta. Tyto zdroje jsou realizovány pomocí abstrakce nad hardwarovými zdroji hosta, na kterém je virtualizace provozována. Bez využití virtualizace je možné mít současně na jednom PC spuštěnou pouze jednu instanci operačního systému, respektive jeho jádra.



Obrázek 2: Znázornění rozdílu tradičního a virtualizačního přístupu [z1]

3.1 Host

Host je samotný fyzický hardware, na kterém chceme virtualizaci provozovat. Provoz virtualizace standardně nevyžaduje žádný specializovaný hardware, protože všechny moderní procesory architektury x86 virtualizaci podporují. Výjimkou mohou být pouze procesory určené pro nízkou spotřebu, jako například rodina procesorů Intel Atom [c2]. Při možnosti volby hosta je v praxi vždy potřeba zvážit jaký typ virtuálních strojů bude na hostu provozován a jaké budou mít požadavky na výkon.

3.2 Hypervisor

Hypervizor je software umožňující virtualizaci, nejrozšířenější hypervizory jsou pak například KVM (zkratka z Kernel-based Virtual Machine), VirtualBox, VMware nebo Hyper-V. Při výběru hypervizoru jsou pro nás mezi nejdůležitější parametry z pravidla patří, cena, požadavky na operační systém hosta, nebo možnost propojení více hostů do clusteru.

	Licence	**Typ**	**Host OS**	**Clustering**
KVM	open source	Hostovaný	Linux	Ano
Virtual Box	open source	Hostovaný	Linux \ Windows \ Mac OS	Ne
VMware Workstation	proprietární	Hostovaný	Linux \ Windows \ Mac OS	Ne
VMware ESXi	proprietární	Nativní	VMware ESXi	Ano
Hyper-V	proprietární	Hostovaný	Windows	Ano

Tabulka 1: Přehled vybraných hypervizorů

3.2.1 Nativní hypervizor

Nativní hypervizor, někdy také označovaný jako hypervizor typu 1 je provozován přímo na hardware hosta bez využití mezivrstvy dalšího operačního systému. Hypervizor má tedy v plné správě veškeré hardwarové prvky hosta a může je volně rozdělovat. Díky ubrání mezivrstvy operačního systému nativní hypervizory dosahují lepšího výkonu než hypervizory hostované. Nativní hypervizory jsou také považovány často za robustnější a bezpečnější řešení, což vychází z toho, že jsou vyvíjené a primárně cílené na firemní serverové nasazení.

3.2.2 Hostovaný hypervizor

Hostovaný hypervizor je spuštěn na operačním systému v podstatě stejně jako ostatní běžné aplikace. Využití mezivrstvy operačního systému s sebou ovšem přináší zvýšení režie. Pokud například aplikace ve virtuálním stroji potřebuje načíst data, je potřeba aby její požadavek šel přes operační systém virtuálního stroje, hypervizor, operační systém hosta na

hardware stejnou cestou se poté vrací i data. Na druhou stranu díky využití mezivrstvy operačního systému je možné hypervizor beze změny provozovat na veškerém hardware podporovaném ze strany výrobce operačního systému. Toto s sebou ovšem může přinést i úskalí v podobě možnosti provozování i firemní virtualizace na běžném spotřebitelském hardware a operačním systému, což s sebou často nese nižší stabilitu a bezpečnost. Zajímavou variantou hypervizoru je KVM, KVM umožňuje virtualizaci přímo v jádře bez další vrstvy operačního systému, takže odpovídá definici nativního hypervizoru. Ovšem na druhou stranu při standardním nasazení KVM získáme kompletně funkční systém Linux a jednotlivé virtuální stroje běží jako běžné linuxové procesy. Z tohoto důvodu osobně preferuji zařazení KVM spíše mezi hostované hypervizory.

3.3 Virtual machine (VM)

Virtual machine je výsledné prostředí pro běh virtualizovaného software. Při klasické plné virtualizaci se toto prostředí vytváří, aby se prezentovalo jako standardní kompletní virtuální počítač, na který se později instaluje operační systém. S virtual machine je ze softwarového pohledu stejná interakce jako s fyzickým strojem. Za předpokladu, že se uživatel připojuje vzdáleně, ani sám nemusí být schopen rozeznat, že se jedná a virtual machine.

3.3.1 Snapshot

Snapshot je technologie zachycení stavu virtualizovaného systému v určitém čase. Snapshoty se nejčastěji využívají jako druh zálohování, ať už se jedná o zálohování pravidelné nebo před provedením operace, která by mohla ohrozit funkčnost a běh virtuálního systému. Existují dva základní typy snapshotu. Je možné vytvářet pouze snapshoty virtuálního disku, což je vhodné převážně pro pravidelné zálohy. Druhou možností je vytvořit snapshot celého virtuálního stroje, tento snapshot poté obsahuje veškerý aktuální obsah virtuálního stroje včetně disku, paměti RAM i stavu procesoru a případných rozšiřujících karet. Tento druh snapshotu nejvíce využijeme právě v případě obnovy po provedení fatální změny.

3.4 Využití virtualizace

3.4.1 Desktopové využití

Tato práce se primárně věnuje využití virtualizace v podnikovém nasazení na serverové aplikace. Ale bylo by vhodné poukázat na skutečnost, že virtualizace je velice užitečný nástroj i pro uživatelské využití na běžných spotřebitelských systémech a hardware.

Mezi nejčastější využití virtualizace běžnými uživateli patří testování. Díky virtualizaci je možné si bez rizika ztráty vlastních dat vyzkoušet například jiné operační. Při využití hypervizoru zaměřených na desktopové využití, je postup vytvoření virtuálního počítače uchopitelný i pro běžné uživatele. Pokud s testováním skončíme je možné prostě celý virtuální stroj zrušit a dále můžeme využívat pouze náš původní os nebo přejít k testování jiného.

Kromě testování celý operačních systémů, může být žádoucí testovat ve virtuálním prostředí i některé aplikace. Tento požadavek může vycházet například z obav ohledně kolizí se současně nainstalovaným software, protože nový software může vyžadovat závislosti na frameworku určité verze, který ale není možné provozovat současně se starou verzí stejného frameworku vyžadovanou původní aplikací. Další věc, která může vést k potřebě instalovat aplikaci do virtuálního prostředí jsou obavy z potencionálního bezpečnostního rizika, které by mohla aplikace přestavovat, pokud by byla nainstalována ve stejném prostředí jako jsou uložena naše citlivá a osobní data.

Další rozšířené využití virtualizace na desktopu je pro provoz aplikací, které jsou určeny pro jiný operační systém, než který je standardně na počítači nainstalován a provozován. V praxi se velice často setkáme s kombinací operačního systému z rodiny GNU/Linux na hostovi a operačního systému rodiny Microsoft Windows ve virtualizačním prostředí například pro využívání kancelářského balíku Microsoft Office, nebo Adobe Photoshop.

3.4.2 Serverové využit

Nyní se dostáváme k způsobu využití virtualizace, kterému se tato práce věnuje. Virtualizace se využívá na serverech hlavně z důvodu možnosti provozu oddělených systémů a aplikací na jednom hardware. Důvodů, proč je toto oddělení vyžadováno je může být hned několik.

Oddělení jako předcházení pádu celé aplikace. je jeden z důvodu proč je vhodné oddělit od sebe jednotlivé systémy aplikace. Vytváří se tím ochrana proti kaskádovému selhání celé

aplikace. Pokud budeme mít například databázový server a aplikační část na jednom systému, jak tomu je třeba u oblíbeného softwarového balíku pro vývoj webových stránek pro os rodiny Microsoft Windows WAMP, a jedna z částí se zachová nestandardně a způsobí pád nejen sebe ale i operačního systému. Přesto přijdeme o část aplikační nehledě na to, jestli by na obsloužení aktuálních požadavků požadovala databázový server či nikoliv. Oproti tomu, pokud oddělíme databázové a aplikační servery a budeme mít každý druh serveru ve více instancích získáme tím princip vysoké dostupnosti. Tento přístup je možný i bez využití virtualizace, ale s využitím virtualizace se stává dostupnější a realizovatelnější.

Virtualizace na serverech je také tažena myšlenkou jednoduší přenositelnosti aplikace. Díky běhu aplikací ve virtualizovaném prostředí je možné, aby programátoři využívali na testování stejné prostředí, jaké se využívá pro ostrý provoz, nezávisle na tom, jaký hardware mají k dispozici pro testování a jaký pro ostrý provoz. Díky přenositelnosti virtualizované aplikace se také začíná stávat irelevantní, jestli bude aplikace běžet v naší serverovně, na pronajatém virtuálním systému nebo v cloudu.

3.5 Výhody virtualizace

V předchozí kapitole jsem se věnovali využití virtualizace na serverech nyní je vhodné představit si jakých výhod se tímto přístupem dá dosáhnout.

Sloučení serverů – díky virtualizaci je možné vzít několik fyzických serverů a převést služby které poskytují na jeden host a několik virtuálních strojů. Tím dojde k úspoře místa, elektřiny, ale také je možné že díky tomu bude možné vyřadit i některé síťové prvky, záložní zdroje a chlazení.

Lepší využití hardware zdrojů – pokud navrhujeme nasazení aplikace na fyzický hardware, je už při pořizování hardware počítat s růstem požadavků aplikace. Proto se vždy vybírá hardware s dostatečnou výkonnostní rezervou, která ovšem nikdy nemusí být využita. Pokud by se jednalo pouze o jeden server není to tak zásadní, ovšem v měřítku aplikace běžící na desítkách serverů by to již mohlo znamenat znatelný ekonomický dopad. S využitím virtualizace můžeme pořízenou potřebnou výkonnostní rezervu snížit na minimum a v případě potřeby přidat další fyzické servery do celého clusteru je jejich zdroje rozdělit mezi systémy virtuální.

Zrychlení a zjednodušení nasazení nových systémů – virtuální systémy je oproti standardním fyzickým systémům mnohem jednodušší ovládat programově. Díky tomu je

možné plně automatizovat kompletní proces instalace a nastavení systému. Další věc, která přispívá k rychlejšímu nasazení, je možnost vytvářet obrazy kompletních systémů. Oproti použití obrazu na fyzické systémy zde nehrozí problémy s ovladači. Neboť v rámci jedné virtualizační platformy máme možnost používat na všech systémech z pohledu ovladačů identický hardware.

Zrychlení obnovy v případě závažných problémů – v předchozím odstavci se řeší jednoduchost nasazení nového systému, pokud toto spojíme s možností zálohování na úrovní celého virtuálního stroje, získáme možnost celou aplikační infrastrukturu v případě potřeby obnovit do funkčního stavu často i v řádu jednotek minut.

3.6 Historie virtualizace

Virtualizace v posledním desetiletí zaznamenala reinkarnaci a výrazný pokrok. Někoho by to dokonce mohlo zmást a mohl by si myslet, že je virtualizace kompletně nová technologie. Opak je však pravdou.

Bylo to už v 60. letech minulého století, kdy na Technické Univerzitě v Massachusetts začal vznikat projekt MAC – Multiple Access Computer, neboli počítač pro vícenásobný přístup. V této době byly veškeré počítače schopny obsloužit pouze jednoho uživatele a jednu úlohu. Nebyl to ještě tak nepřekonatelný problém, protože všechny počítače byli skoro výhradně využívány vždy pouze úzkým okruhem lidí a nebyl tedy problém se domluvit.

O projekt MAC se začala zajímat americká firma IBM, na jeho základě vytvořila vlastní počítač CP-40, který je považovaný za první počítač opravdu využívající virtualizaci na základě sdílení procesorového času mezi procesy. Tento systém se ovšem nikdy nedostal do komerčního prodeje, ale byl využíván pouze interně pro vývoj v samotném IBM. Až v roce 1967 jej následoval model CP-67, který již byl uveden na komerční trh [c3].

Další pokrok na poli virtualizace se odehrál o skoro 40. let podeji, kdy přišel americký startup nazvaný VMware. VMware jako realizovali dynamický překlad privilegovaných instrukcí na neprivilegované instrukce, jejich provedení hypervizorem a vrácení výsledku původnímu volání instrukce. Díky tomu začalo být možné virtualizovat celé systémy bez ohledu na jejich architekturu a operační systém. Díky tomu se VMware stal jedničkou na trhu, kterou setrval dodnes, kdy drží stále okolo 45 % celého trhu s virtualizací [c4].

Do roku 2007 [c5] byl VMware prakticky bez konkurence, tohoto roku ovšem v rámci Linuxového jádra verze 2.6.20 přibyla nativní podpora virtualizace nazvaná KVM. Díky tomu

se možnost virtualizace dostala na všechny počítače s operačním systémy rodiny GNU/Linux. Na vývoji KVM se stejný rok také začala podílet firma Red Hat, která patří mezi největší přispěvatele do Linuxového jádra a nabízí vlastní Linuxové distribuce a jejich podporu. Dnes je KVM začleněno do komplexního systému pro virtualizaci oVirt, který je rozšířený i do firemní sféry kde se často používá pro stavbu privátních cloudových řešení.

Ve stejném roce přišla také Německá firma Innotek GmbH na trh s vlastním open source řešením zaměřeným převážně pro desktopové použití s názvem VirtualBox. Jednalo se o první open source řešení právě pro desktopové použití. O rok později byl Innotek GmbH koupen americkou firmou Oracle pod jejíž hlavičkou vývoj VirtualBoxu pokračuje dodnes. VirtualBox se díky své jednoduchosti a skutečnosti, že je distribuovaný zdarma stal nejpopulárnějším řešením pro desktopovou virtualizaci [c5].

3.7 Příklad podnikového nasazení

Tento příklad vychází z vlastních poznámek k reálné konsolidace infrastruktury, provedené v rámci praxe, bylo změněno pouze jméno firmy a zaokrouhleny procenta průměrného využití zdrojů a spotřeby na celé desítky.

Mějme firmu FUTO s.r.o, tato firma nyní využívá různé aplikace na pěti fyzických serverech popsaných v následující tabulce:

Shrnutí hardware zdrojů	**Průměrné využití zdrojů**	**Velikost**	**Průměrná Spotřeba**	**OS**	**Role**
4 vlákna cpu, 8 GB RAM, 320 GB raid1	20 % CPU 60 % RAM 50 % Disku	2U	350 W	Windows Server 2012	Doménový Kontrolér a Sdílení disků
8 vláken cpu, 24 GB RAM, 120 GB raid1	30 % CPU 80 % RAM 70 % Disku	4U	530 W	Windows Server 2012	Databázový server
4 vlákna cpu, 4 GB RAM, 120 GB raid1	20 % CPU 50 % RAM 60 % Disku	2U	230 W	Windows 7	Účetní aplikace
4 vlákna cpu, 8 GB RAM, 60 GB raid1	10 % CPU 20 % RAM 20 % Disku	2U	170 W	Centos 7	Webový server

4 vlákna cpu, 4 GB RAM, 320 GB raid1	10 % CPU 60 % RAM 10 % Disku	2U	240 W	Centos 7	WiFi controller a Radius server

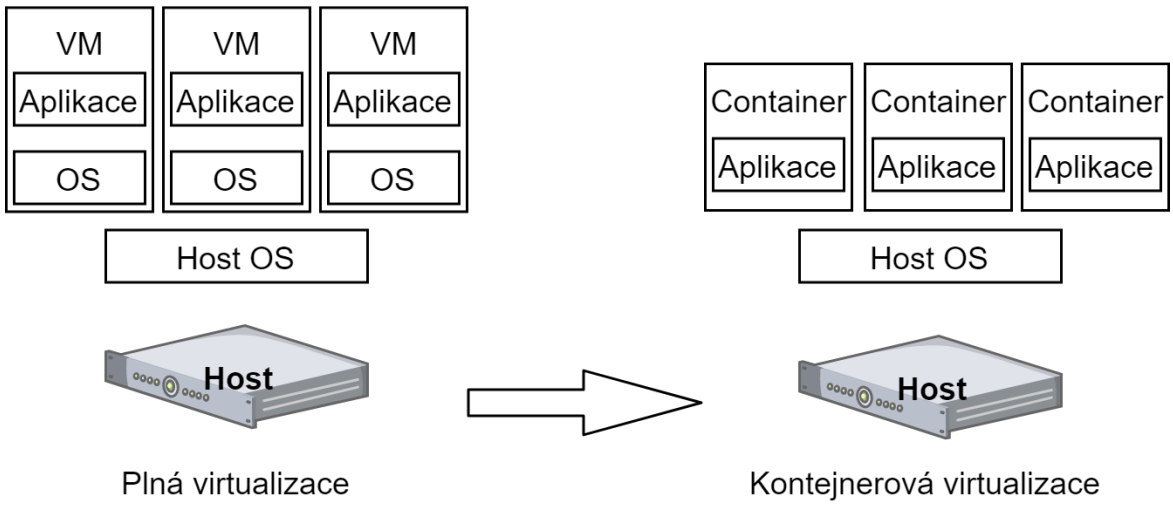
Tabulka 2: Seznam a vlastnosti serverů aplikací před konsolidací

Firma bez využití virtualizace nemůže jednoduše sloučit veškeré role na jednom serveru, a to hlavně z důvodu, že jednotlivé aplikace vyžadují různé operační systémy. Ale ani prosté sloučení aplikací vyžadujících stejný operační systém by nebylo vhodné, a to jak z bezpečnostního hlediska, tak z hlediska závislostí aplikací na rozdílné verze a nastavení frameworků a ostatních balíčků. Firma FUTO s.r.o se tedy po konzultaci s externí firmou rozhodla pro přechod na virtualizační platformu VMware ESXi, nákup nového hardware pro hosta a následný odprodej původního hardware.

Byla tedy provedena konsolidace na jeden nový host. Nový host má 16 vláken CPU, 64 GB RAM a 1Tb diskového prostoru v RAID 5. Spotřeba klesla z 1520 W na pouhých 370 W samotného nového hostu. V racku nyní taky přibylo 10U místa, protože nový host zabírá pouze 2U. Průměrné využití nového hosta se pohybuje kolem 30 % CPU a 50 % RAM. Nový host tedy není plně využit, protože se počítá s růstem firmy a přidáním nových systémů.

4 Kontejnerová virtualizace

Další možností virtualizace je kontejnerová virtualizace. Kontejnerová virtualizace posouvá myšlenku virtualizace od virtualizace celých serverů k virtualizaci pouze operačních systémů. Při využití kontejnerové virtualizace jsou oproti standardní virtualizaci, kde je oddělen jak uživatelský prostor, tak prostor jádra, odděleny pouze prostor uživatelský. Prostor jádra je tedy sdílen jak s operačním systémem jádra, tak s ostatními virtualizovanými systémy. Tento přístup s sebou přináší hlavní nedostatek kontejnerové virtualizace, není možné virtualizovat jiný operační systém nebo jinou architekturu, než je používán na hostu.



Obrázek 3: Znázornění rozdílu plné a kontejnerové virtualizace [z1]

4.1 Výhody kontejnerové virtualizace

Kontejnerová virtualizace spolu nese veškeré výhody klasické plné virtualizace. Díky odstranění mezivrstvy operačního systému ve virtuálním systému má však znatelný výkonnostní nárůst oproti virtualizaci klasické. Také jsou zde sníženy zdroje potřebné k režii. V praxi je zde nejvíce znatelný úbytek využití paměti RAM. Při mém testování CPU výkonu pomocí programu *sysbench* s parametry *--test=cpu --num-threads=4 --cpu-max-prime=9999 run* jsem zaznamenal u plné virtualizace pomocí KVM ztrátu výkonu pohybující se průměrně kolem 11 %. Při využití LXC kontejneru byla tato ztráta průměrně pouze 4 %.

4.2 Historie kontejnerové virtualizace

Ač by se z úvodu této kapitoly mohlo zdát, že se jedná v případě kontejnerové virtualizace o novou věc, ve skutečnosti její kořeny sahají už do roku 1979 [c6], kdy byl poprvé představen tehdy ještě na systému unix příkaz chroot. Chroot je zkratkou z changed root, neboli změněný root a to je také podstata toho co tento příkaz dělá [c7]. Příkaz vytvoří pro další příkaz a jeho potomky nový umělý kořenový adresář na původním souborovém systému. Jednalo se o první systém na izolaci procesů a prvotní krok ke kontejnerové virtualizaci.

Další pokrok v kontejnerové virtualizaci nastal až zhruba po 20 letech kdy bylo v roce 2000 uvedeno Jails pro operační systém FreeBSD [c8]. Jails oproti chrootu přináší dvě zásadní novinky. Je zde zvýšena bezpečnost, virtuální systém nyní má kompletně oddělené uživatele od hosta. Takže v případě, že by útočník získal root přístup k virtuálnímu systému, stále by neměl mít možnost zasáhnout do systému hosta. Druhou novinkou je možnost přidělení alokovaného síťového rozhraní virtuálnímu systému [c9].

V roce 2001 přichází Linux-VServer [c10], z pohledu vlastností a možností VServer nabízí podobné možnosti jako FreeBSD Jails a přináší je na systém Linux. Pro provoz VServeru na Linuxovém systému je potřeba nainstalovat upravené Linuxové jádro s jeho podporou [c11]. I přesto, že poslední stabilní vydání vyšlo před 9 lety v roce 2008, jedná se stále o používaný systém kontejnerové virtualizace.

Za zkratkou LXC z roku 2009 [c12] se skrývá LinuX Containers a jedná se o první platformu pro kontejnerovou virtualizaci již ve formě jakou známe dnes. Nevyžaduje žádné dodatečné úpravy jádra hosta. A přináší podporu virtualizace namísto pouze alokace síťových rozhraní. Je tedy možné definovat i routy pro jednotlivé virtuální systémy [c13]. Oblibu LXC je možné demonstrovat na tom, že je dodnes součástí, jako varianta ke klasické virtualizaci, i oblíbeného virtualizačního systému Proxmox VE [c14].

4.3 “Mazlíček nebo Stádo”

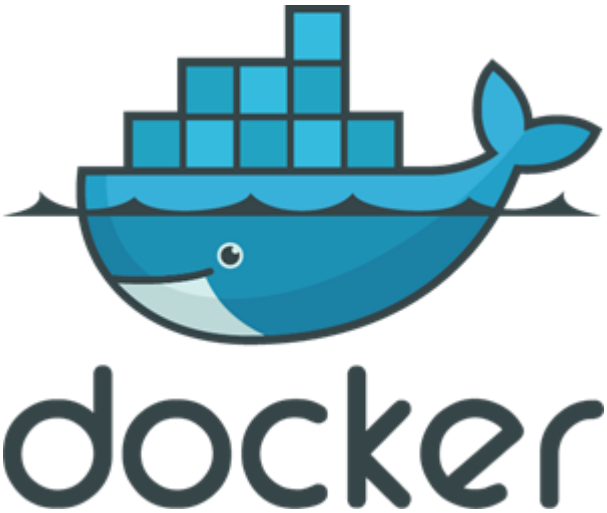
Moderní kontejnerová virtualizace a Docker přesně reaguje a odpovídá na myšlenku, která se v odborném slangu označuje jako “Mazlíček nebo Stádo”. Celá tato myšlenka je o zásadní změně přístupu k systémům. Původní přístupu k systému je jako ke svému “mazlíčkovi”, o kterého se staráme a víme o něm první poslední a pokud něco přestane fungovat, řešíme problém přímo na systému. Tento přístup ovšem nestíhá reagovat a celkově je v nesouladu s trendem poskytování aplikací, infrastruktury a systémů jako službu.

Proto se zavádí nový přístup, a to přístup k systémům jako ke stádu. To znamená že ve výsledku mě už nezajímají tolik jednotlivé systémy, ale podstatné je pro mne, aby stádo jako takové plnilo svou funkci. Pokud dojde k problému na jednom ze systémů, je zde žádanější a rychlejší prostě celý systém zrušit a nahradit ho novou instancí stejného tipu. A to je zároveň nejsilnější myšlenka přístupu k serveru jako ke stádu, je potřeba počítat s tím, že systémy jako jednotlivci jsou postradatelní a rychle nahraditelní. Proto by neměli nést žádná data, jejichž ztráta by ohrozila celou aplikaci, "stádo".

Osvojení a zavedení tohoto přístupu ovšem není pouze v režii správců počítačových systému, ale je potřeba aby pro něj byly optimalizované aplikace již od jejich prvotního návrhu.

5 Docker

Docker je software platforma pro kontejnerovou virtualizaci od stejnojmenné americké společnosti. Docker byl poprvé veřejně vydán 23.3.2013 [c15] ve verzi 0.1.0. K dnešnímu dni je dostupná verze 17.12.1. Během vývoje také došlo k změně aplikace Dockeru jako takové. Při vydání byl označován pouze jako Docker a toto označení mu zůstalo do verze 1.13 vydané v vydané na začátku roku 2017. Koncem roku 2017 došlo k rozhodnutí přejmenovat samotnou aplikaci a rozdělit její verze na komunitní a enterpirse verzi a také změnit číslování verzí. Z tohoto rozhodnutí vyšly tedy verze Docker CE a Docker EE, respektive comunity edition a enterpirse edition.



Obrázek 4: Logo aplikace Docker [z2]

5.1 Komponenty Dockeru

Před dalším popisem Dockeru by bylo rozhodně vhodné popsat jeho jednotlivé části.

Docker Engine je hlavní částí Dockeru, jedná se o samotný hypervizor druhého typu, je tedy hostovaný nad operačním systémem. Docker Engine je možné provozovat na všech třech nejrozšířenějších operačních systémech. Pro enterprise podporu je ovšem nutný jeho provoz na operačních systémech rodiny Microsoft Windows nebo rodiny GNU\Linux. Zajišťuje vytvoření, běh a správu samotných kontejnerů s virtuálními systémy.

Kontejner je v rámci Dockeru a kontejnerové virtualizace označení virtuálního systému. Využívá se zde i přirovnání k reálnému lodnímu kontejneru, neboť stejně jako lodní kontejner i ten virtuální by měl mít jednotné předem definované a ustanovené parametry, rozhraní. A měl by být bez problémů kompatibilní se všemi Docker hosty nezávisle na jejich hardware.

Využití Dockerfile je pro mnohé odborníky převážně z řad programátorů jedna z nejpokrokovějších věcí, které Docker zavedl. Díky tomu dostali do rukou silný nástroj pro definování systému pomocí kódu. V praxi je tedy mnohem jednoduší udržet jednotné prostředí jak pro vývoj a testování, tak pro ostré nasazení do praxe.

Docker veškeré spuštěné kontejnery odvozuje od jejich snaphsotů, které jsou zde nazývány Image. Jedná se o diskové snapshoty virtuálních systémů. Tyto image je možné vytvářet jak pomocí definice v Dockerfile, tak jako diskový snapshot již běžícího kontejneru. Druhý zmiňovaný způsob je ovšem samotnými autory Dockeru silně nedoporučován pro jakékoliv jiné použití, než je testování a debuggování.

Veškeré obrazy Docker získává z registru, tento registr je při holé instalaci Dockeru realizován pouze adresářovou strukturou na souborovém systému hosta anebo využíváním oficiálního Docker registru. Realizace vlastního sdíleného registru je ale vcelku je jednoduchá. Stačí pouze použít oficiální obraz registru a připojit mu složku výše zmíněnou adresářovou strukturou na souborovém systému hosta. Poté je možné používat obrazy z ní i na ostatních Docker hostech, které jsou nastaveny, aby s tímto registrem komunikovali.

5.2 Výhody Dockeru

Na prví pohled by se mohlo zdát, že je Docker pouze další platforma pro kontejnerovou virtualizaci, ale Docker jako první zavádí a popularizuje zásadní a revoluční myšlenky.

Princip kontejneru je sice už popsán v předchozí kapitole, ale Docker tento koncept rozšiřuje vlastní myšlenku. V souladu s filosofií Dockeru by měl každý kontejner obsahovat pouze jeden proces. Respektive by měl plnit pouze jednu logickou funkci. Pokud to popíšu v praxi na příkladu jednoduché webové prezentace to znamená, že bude běžet zvlášť kontejner pro fronted, databázi, frontu, a různé backendové služby.

Všechny objekty Dockeru by měly být jednoduše recyklovatelné, proto je i k přímému spuštění kontejneru potřeba obraz, otisk virtuálního systému. Docker spravuje obrazy po vrstvách a využívá odvozování obrazů od jiných obrazů. Tím pádem, pokud používáme v našem prostředí kontejnery založené všechny na stejném výchozím obrazu, budeme mít základní obraz uložený pouze jednou a poté budeme mít pouze rozdíly různých kontejnerů oproti výchozímu obrazu. Protože obrazy neobsahují jádro je možné také často narazit pouze na kontejnery, které mají pouze jednotky megabajtů.

K tvorbě obrazů Docker využívá již výše zmíněný Dockerfile. Jedná se o textový soubor definující obraz, z jakého obrazu vychází, jaký software se má instalovat, jaké soubory se mají zakomponovat do obrazu a jaké porty má výsledný kontejner vystavit do Dockeru. Jedná se tedy o přístup "systém as a code", kdy je celý systém reprezentovaný pouze jako kód.

Z myšlenky recyklovatelnosti vychází požadavek na bezstavovost samotných kontejnerů. Reálně to znamená, že by kontejner neměl obsahovat žádná data, jejichž ztráta by nepříznivě ovlivnila chování celé aplikace. Pokud je potřeba aby kontejner měl přístup k datům, která mají být persistentní, je nutné použít připojených dat hosta. Toto je řešeno buďto přímým připojením složky nebo souboru přímo ze souborového systému hosta. Nebo pomocí vytvořením takzvaného Volume a jeho následné připojení kontejnerům.

Aby bylo možné realizovat všechny předchozí myšlenky disponuje Docker systémem orchestrace také nazývaným Docker Compose. Využívají se zde soubory yamlové syntaxe a popisuje se zde celá aplikace, v tomto případě nazývaná služba. V těchto souborech je možné popsat všechny používané kontejnery aplikace, definovat sítě, volume. Tak jak Dockerfile je "system as a service" tyto soubory jsou "infrasturcture as a service" neboli infrastruktura jako kód. Docker se poté stará o to aby služba odpovídala své yamlové definici i v případě výpadku některých z kontejneru.

5.3 Životní cyklus kontejneru v Dockeru

Každý kontejner v Dockeru začíná svou definicí pomocí DokcerFile. Jelikož se jedná o definici systému pomocí yaml kódu prolínají se při práci na Dockerfile kompetence jak systémových administrátorů a architektů, tak programátorů. Po definování Dockerfile je tento soubor pomocí nástrojů v Docker Engine překompilován na image, obraz, vužívá se zde skládání a verzování jednotlivých částí obrazu, takže není nutné při drobných úpravách nutné měnit celý obraz systému. Po úspěšném sestavení je obraz buď uložen pouze v lokálním, pokud se jedná pouze o lokální testování, anebo je odeslán do vzdáleného registru. Pokud Docker Engine dostane požadavek na spuštění nového kontejneru, prohledá nejprve lokální registr, jestli obsahuje požadovaný systémový obraz. Pokud není obraz nalezen v lokálním registru, pokračuje s prohledáním vzdálených registrů. Mezi ty při standardní instalaci patří pouze oficiální registry hub.docker.com. Po nalezení obrazu je tento obraz zaveden do paměti a je z něj spuštěn kontejner, který je dále prezentován uživateli.

5.4 Docker Swarm

Docker Swarm je platforma pro spojení několika hostů s Dockerem do clusteru se všemi jeho výhodami. Docker Swarm začalo jako samostatný projekt běžící jako jednotlivé Dockerové kontejnery, které běžely v privilegovaném režimu, aby mohli ovlivňovat chování hosta a samotného Dockeru. Celý systém byl složen z kontejneru pro každého hosta, kontejnerů konzulů pro jejich řízení a kontejnerů pro sdílení klíčových hodnot pomocí etcd. Od verze 1.12.0 je Docker Swarm přímo součástí instalace Docekru.

Docker Swarm přináší do Dokceru možnost vytvářet z několika samostatných hostů cluster a vyžívat jeho výhody. Zaměříme se teď na pár nejzákladnějších myšlenek a výhod clusterového řešení pomocí Docker Swarm.

Díky zapojení do Docker Swarm rozšíříme naše možnosti škálování aplikace do boku, tedy přidáním její replik. Budeme totiž schopni řídit veškeré repliky z jednoho centrálního místa a jejich distribuci na jednotlivé hosty bude zařizovat Docker Swarm.

Díky zapojení do clusteru pomocí Docker Swarm získáme možnost automatického udržení vysoké dostupnosti jak jednotlivých kontejnerů, tak celé naší aplikace. V případě selhání některého z hostů jsou totiž kontejnery, které na něm běžely opět nastartovány na jiném hostu a tím se obnoví jejich původní počet a vysokou dostupnost.

Myšlence udržení vysoké dostupnosti napomáhá také fakt, že Docker Swarm využívá decentralizovaný design, kdy veškeré jeho řízení a koordinace je v případě korektní instalace rozdělena mezi několik manažerských hostů. Díky tomu zde není žádný jednotný slabý bod, jak by tomu mohlo být v případě, že by kontrolou nad celým Docker Swarm měl pouze jeden bod.

Aby bylo možná komunikace mezi jednotlivými kontejnery napříč Docker Swarm, je Docker Swarm vybaven overlay síťovou vrstvou. Využití této vrstvy nám umožňuje využívat přístup definování sítě jako kusy kódu a v případě potřeby je programově ovlivňovat jejich chování a strukturu.

6 Raspberry Pi cluster

Jak již bylo řečeno cílem práce je vytvořit přenosný nástroj, na kterém je možné prezentovat vlastnosti kontejnerové virtualizace a clusteru širší odborné veřejnosti. Z toho vychází samotný návrh clusteru a volba jeho komponent. Kontejnerovou virtualizaci by samozřejmě bylo možné prezentovat na clusteru sestaveném z výkonných serverů umístěných v datacentru. Tento způsob prezentace má ale několik zásadních nevýhod, kvůli kterým padlo rozhodnutí pro tuto práci využít mikropočítače Raspberry Pi. Prvním problémem by bylo prezentace škálování, balancování zátěže a vysoké dostupnosti. Pro všechny tyto ukázky by musela být do jisté míry použita představivost publika, dostali bychom se tedy do situace "nyní jsem vypnul jeden z hostu v datacentru a vidíte stále to funguje". Při použití Raspberry Pi je možné díky jejich přenosnosti tuto demonstraci provádět přímo na místě. Druhou výhodou, ač se to na první pohled nezdá je nízký výkon Raspberry Pi. Díky tomu je časový rozdíl mezi klasickou instalací prezentované aplikace a jejím nasazením pomocí Dockeru ještě znatelnější.

6.1 Raspberry Pi

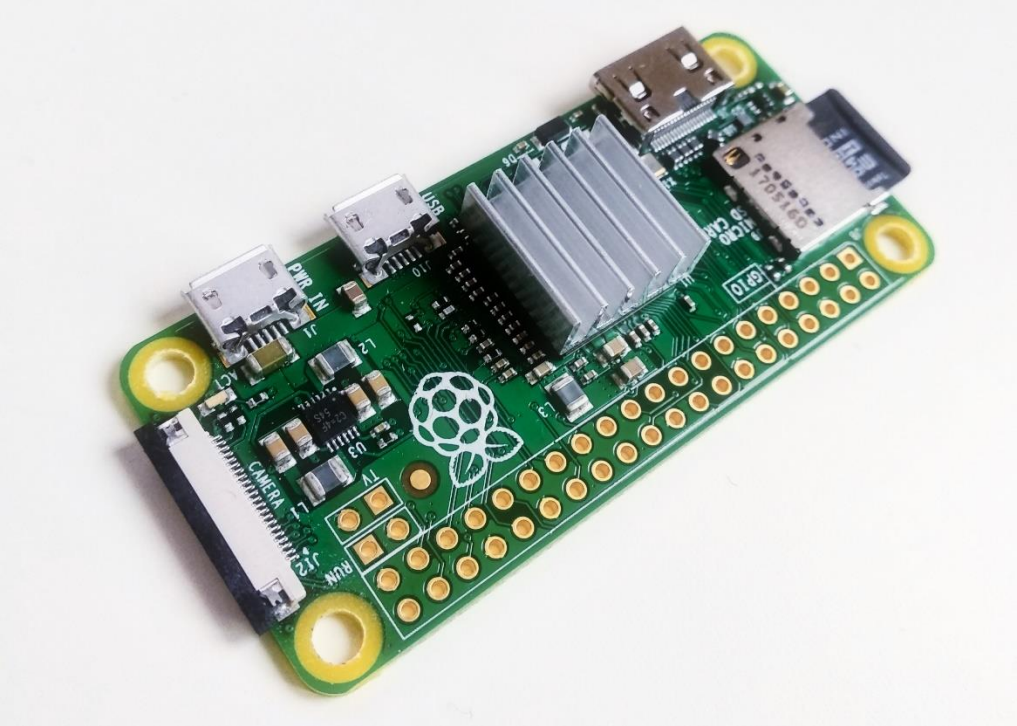
Při volbě Raspberry Pi jsem měl volbu mezi 3 aktuálně prodávanými modely. Jednalo se o Raspberry Pi 3 Model B, Raspberry Pi Zero a Raspberry Pi Zero W. Základní vlastnosti jednotlivých modelů, které ovlivňovali moje rozhodování jsou popsány v následující tabulce.

	Raspberry Pi 3 Model B	Raspberry Pi Zero	Raspberry Pi Zero W
CPU	1.2GHz 64-bitový čtyř jádrový ARM Cortex-A53	1GHz jedno jádrový ARM1176JZF-S	1GHz jedno jádrový ARM1176JZF-S
Velikost RAM	1 GB	512 MB	512 MB
Počet USB portů	4	1	1
Wifi	Ano	Ne	Ano
Ethernet konektor	Ano	Ne	Ne
Cena na RPishop.cz	949 Kč	157 Kč	313 Kč

Tabulka 3: Porovnání aktuálně prodávaných modelů Raspberry Pi

Technické parametry a cena jsou převzaty pouze z RPishop.cz z důvodu, že se jedna o jediného oficiálního prodejce pro český a slovenský trh uznaného Raspberry Pi Foundation [c16]. Ceny jsou platné ke dni 16.1.2018.

Z tabulky jasně vyplývá výrazně lepší hardware vybavenost Raspberry Pi 3 Model B. Je také ale vidět, že Raspberry Pi 3 Model B se prodává za šestinásobek ceny Raspberry Pi Zero. Jelikož jsem pro stavbu clusteru potřeboval zařízení pět byl jsem z ekonomických důvodů nucen vybrat variantu Raspberry Pi Zero. Výsledný cluster tedy bude disponovat 5 procesorovými jádry a 2560 MB paměti RAM.



Obrázek 5: Fotka Raspberry Pi Zero s instalovaným pasivním chladičem [z1]

6.2 Ostatní komponenty

Cluster samozřejmě není složen pouze ze samotných Raspberry Pi Zero, pro jeho funkčnost bylo zapotřebí ještě další příslušenství, které překvapivě nakonec hodnotou přesahuje hodnotu samotných pěti Raspberry Pi.

6.2.1 SD Karty

Jak již bylo řečeno v kapitole věnující se Dockeru, velikost jednotlivých obrazů a kontejnerů je relativně nízká. Velké množství dat také ušetří instalace Raspbianu bez grafického rozhraní, tedy ve verzi Lite. Proto jsem došel k závěru, že kapacita karet 8 GB bude dostatečná. Dalším důležitým parametrem byla volba rychlostí zápisu a čtení SD karty. SD karty jsou podle rychlostí standardně řazeny do rychlostních tříd. Zvolil jsem tedy SD karty Kingston MicroSDHC 8 GB Class 10 UHS-I Industrial Temp, jak je z názvu patrné tato karta je marketována jako rychlostní třída 10. Tato karta má také oproti standardním kartám větší rozsah pracovních teplot až do 85 °C, měl jsem obavy z vyšších teplot při provozu Raspberry v malé vzdálenosti od sebe. Také cenový rozdíl mezi standardní a Industrial Temp verzí byl zanedbatelný.

6.2.2 Switch/Router

Jako router jsem pro cluster zvolil TP-Link TL-WR741ND v1, protože tento router podporuje na Linuxu založený firmware DD-WRT. Tento firmware mimo jiné nabízí možnost připojení WAN portu routeru do switche s ostatními LAN porty, takže jsem mohl jednoduše připojit všech 5 Raspberry Pi. Připojení clusteru k internetu je poté realizováno pomocí Wi-Fi, kdy je router konfigurován jako klient. Pro přístup ze zbytku naší sítě k jednotlivým Raspberry Pi v clusteru jsou jak na routeru clusteru, tak na hlavním routeru naší sítě nastaveny statické routy mezi sítěmi.

6.2.3 Síťové karty

Raspberry Pi Zero není osazeno síťovou kartou. Bylo tedy potřeba pro Raspberry Pi Zero dokoupit USB síťové karty. Během výběru karet jsem nalezl síťové karty přímo s konektorem micro USB s podporou USB OTG. Karty označené Kebidu Micro USB Network Card mají i nativní podporu v operačním systému Raspbian a proto se zdály pro mé řešení naprosto dokonalé.

Při zapojení do clustery jsem ovšem narazil na problém. Karty sice signalizovaly aktivitu, ale router, ke kterému byli připojené přestal reagovat jak na webovém rozhraní, tak na icmp ping. Po detailnějším prozkoumání jsem zjistil, že všech 5 zakoupených karet má stejnou hardwarovou MAC adresu. Tento problém jsem vyřešil pomocí skriptu, který před aktivací USB karty jí nastaví předem definovanou MAC adresu podle hostname zařízení. Skriptu se dále podrobněji věnuji v kapitole instalace a nastavení operačního systému.

6.2.4 USB hub

Tento USB hub bude sloužit pouze pro napájení Raspberry Pi Zero a nebude se využívat pro žádné datové přenosy, Proto pro výběr USB hubu byly důležité pouze dva parametry, minimální počet portů a možnost externího zdroje napájení. Zvolil jsem nakonec 7 portový Akasa Connect 7+. Je schopný poskytovat až 500 mA [c17] na všech portech zároveň. 500 mA je dostatečné pro stabilní běh Raspberry Pi Zero i při plné zátěži vyžaduje 240 mA [c18].

6.2.5 Cenová kalkulace

V následující tabulce jsou shrnuty veškeré mé náklady na pořízení samotných Raspberry Pi a ostatních komponent clusteru. Tabulka by měla sloužit především případným zájemcům o replikování mého projektu.

Název	**Cena (Kč)**	**Doprava (Kč)**	**Cena celkem (Kč)**
5 x Raspberry Pi Zero	157	45 za kus	1010
5 x Síťová karta	63	-	315
5 x SD karta	221	-	1105
5 x microUSB kabel	54	-	272
5 x Patch kabel	36	-	182
1 x USB Hub	497	-	497
1 x Router/Switch	517	-	517
Celkem:			**3898**

Tabulka 4: Výsledná cenová kalkulace

V tabulce je také vidět, že cena za dopravu u Raspberry Pi Zero je uvedena za kus. Kvůli velkému zájmu a nedostatečné produkci bylo Raspberry Pi Foundation přinuceno ke kroku vydání omezení prodeje na 1ks Raspberry Pi Zero a Raspberry Pi Zero W na zákazníka. Bylo tedy potřeba objednat 5 potřebných kusů na různé členy mé rodiny.

6.3 Sestavení

Pro účely testování jsem celý cluster provozoval bez jakékoliv podpůrné struktury pouze rozložený po stole. Tento přístup by byl ovšem pro jakoukoliv přenosnost a prezentování krajně nevhodný.

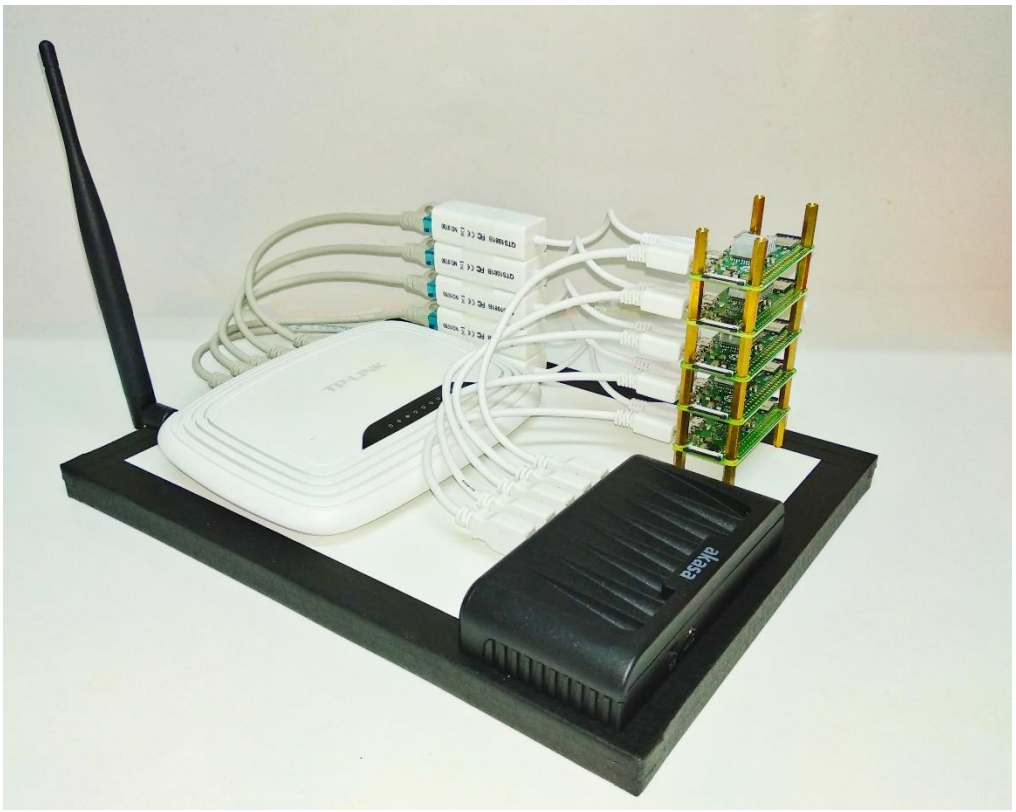
Rozhodl jsem se tedy, že celý cluster spolu se síťovými a napájecími prvky bude vhodné umístit na platformu z dřevěné desky. Po modelování možností rozložení clusteru na různé varianty dřevěné platformy jsem došel k závěru, že bude vhodné jednotlivé Raspberry Pi umístit

vertikálně nad sebe pomocí distančních šroubů a využití dřevěné platformy o rozměrech 230x300x20mm.

Kvůli špatnému pochopení oficiální mechanické dokumentace jsem ovšem pořídil šrouby rozměru M3x20mm místo M2.5x20mm [c19]. Musel jsem tedy převrtat montážní díry na jednotlivých Raspberry Pi.

Samotný cluster je do dřevěné platformy zapuštěný 5 mm a uchycen pomocí epoxidového lepidla. K platformě jsou poté napájecí a síťové prvky přichyceny pomocí roztaveného plastového polymeru.

Při umístění jednotlivých Raspberry Pi nad sebe s mezerou 20 mm jsem začal pozorovat zvýšení provozních teplot jejich procesorů. Teploty se pohybovali kolem hranice 60 °C při zátěži. Rozhodl jsem se tedy dokoupit pasivní chladiče o rozměrech 14x14x6mm. Tyto chladiče jsou připevněny na jednotlivé procesory pomocí samolepících chladících podložek. Díky této úpravě klesly teploty při plné zátěži o 15 °C a nyní se pohybují okolo 45°C.



Obrázek 6: Fotka finálního clusteru bez napájecích zdrojů [z1]

6.4 Instalace a nastavené operačního systému

Jako operační systém pro cluster je zvolena linuxová distribuce Raspbian verze 9 s kódovým označením Stretch. Tento sytém vychází z hojně rozšířené distribuce Debian a jedná se o její upravenou verzi speciálně pro Raspberry Pi. Hlavním rozdílem oproti standardnímu Debianu pro procesory rodiny ARM je, že Raspbian je schopen využívat hardwarový numerický koprocesor Raspberry Pi pro akceleraci numerických operací [c20]. K volbě tohoto systému samozřejmě také přispěla oficiální podpora ze strany samotné Raspberry Pi Foundation [c21]. K práci s clusterem je převážně používán SSH přístup, takže bude stačit použít pouze Lite variantu systému, která neobsahuje grafické rozhraní. Veškeré příkazy v této kapitole jsem prováděl s právy super uživatele root.

6.4.1 Instalace

Z oficiálních stránek [c22] jsem stáhl a rozbalil diskový obraz systému. Před samotným zápisem systému na SD kartu je vhodné ověřit hodnotu jeho SHA-256 hashe a porovnat ji s hodnotou uvedenou na výše odkazovaných oficiálních stránkách. Tím si ověřím, že se obraz stáhl a rozbalil bez změny oproti originálu. Jelikož jsem instalaci prováděl na operačním systému Fedora z rodiny operačních systémů GNU/Linux kontrolu jsem provedl pomocí příkazu sha256sum [c23]:

```
sha256sum ~/Downloads/raspbian.img
```

Dále jsem pokračoval samotným zápisem obrazu na SD kartu. Využil jsem k tomu příkaz dd [c24]:

```
dd if =~/Downloads/raspbian.img | pv | dd of=/dev/sda
```

Po dokončení příkazu bylo potřeba kartu odpojit a znovu připojit, aby se načetly změny v tabulce oddílů. Po znovu připojení se už načetly oba nové oddíly a mohl jsem pokračovat v po instalačních krocích.

6.4.2 Po instalační kroky

Dva nově vytvořené oddíly na SD kartě jsou boot a rootfs, boot obsahuje soubory spojené se startem systému a rootfs obsahuje samotný souborový sytém Raspbianu. Prvotní a nejdůležitější krok po instalaci bylo povolení SSH pro vzdálené připojení. Pro povolení SSH při startu jsem vytvořil soubor “ssh” v kořenové složce boot oddílu na SD kartě.

Dále jsem se přesunul k oddílu rootfs. Protože jsem během prvotního testování funkčnosti komponent narazil na problém, že všechny použité síťové karty pro jednotlivá Raspberry Pi mají stejné MAC adresy, vytvořil jsem tedy skript, který problém vyřeší. Skript na základě hostname jednotlivých Raspbery Pi nastaví pro připojenou síťovou kartu jednu z předem definovaných MAC adres.

```
#!/bin/sh
case $(cat /etc/hostname) in
        'rpi01')
            ip link set dev eth0 address 44:ab:41:33:b4:51
            ;;
        'rpi02')
            ip link set dev eth0 address 44:ab:41:33:b4:52
            ;;
        'rpi03')
            ip link set dev eth0 address 44:ab:41:33:b4:53
            ;;
        'rpi04')
            ip link set dev eth0 address 44:ab:41:33:b4:54
            ;;
        'rpi05')
            ip link set dev eth0 address 44:ab:41:33:b4:55
            ;;
esac
exit 0
```

Tento skript jsem umístil do složky /etc/network/if-pre-up.d a pojmenoval jej clusterMAC. Umístěním skriptu do této složky jsem docílil toho, že skript se spustí ještě před samotnou aktivací rozhraní a předejde se tím problému více stejných MAC adres v rámci jedné sítě. Aby tento skript mohl správně určit jakou MAC adresu přidělit, bylo ještě potřeba definovat hostname Raspbery Pi. Upravil jsem tedy /etc/hostname a /etc/hosts aby obsahovali správné hostname.

Když jsme měl připravené SD karty pro všechny Raspberry Pi, zapojil jsem je do clusteru, aby hostname byli očíslované od nejspodnějšího Raspberry Pi vzhůru a připojil jsem USB hub do napájení. První start byl delší, protože se během startu provádí ještě automatické dodatečné rozšíření oddílu rootfs a jeho souborového systému, aby byla využita celá kapacita SD karty. Po ověření dostupnosti ssh a možnosti přihlášení jsem mohl pokračovat s instalaci Dockeru.

7 Instalace Docker

Po instalaci, nastavení operačního systému a před samotným začátkem instalace nového software jsem podle doporučených linuxových praktik aktualizoval data o depozitářích a nainstaloval nejnovější dostupné verze všech instalovaných balíčků.

```
apt update
apt upgrade -y
```

Když jsem měl všechno aktuální, využil pomocí následujícího příkazu jsem oficiální instalační skript pro instalaci Dockeru.

```
curl -sSL https://get.docker.com | sh
```

Tyto poslední dva kroky, které se už prováděli pomocí hardwarových zdrojů Raspberyy Pi ovšem názorně demonstrovaly schopnosti Raspberry Pi, neboť jejich zpracování trvalo průměrně 18 minut.

Když byla instalace dokončena provedl jsem kontrolu funkčnosti Dockeru zatím bez jejich propojení do Docker Swarm pomocí příkazu:

```
docker run -d -p 80:80 php:7.0-apache
```

Tento příkaz stáhl obraz Dockrového kontejneru aplikace Apache s PHP ve verzi 7.0 z oficiálního repozitáře Docker Hub. Parametrem -p je nastaveno že port 80 z kontejneru bude propagován na portu 80 na samotném hostu. Po stažení a spuštění kontejneru jsem zadal adresy jednotlivých Raspberry Pi do webového prohlížeče a ověřil jsem, že se mi opravdu na všech Raspberry Pi načte výchozí stránka Apache s textem “It works”.

7.1 Nastavení Docker Swarm

Po ověření funkčnosti Dockeru na všech Raspberry Pi jsem přešel k instalaci Docker Swarm. Pro začátek bylo potřeba rozhodnout jakou budu používat strategii rozdělení v clusteru a množství Swarm managerů, které budou používat. Swarm manager funguje stejně jako ostatní Docker hosti, ale je rozšířen o pravomoci provádět změny v rámci celého clusteru, spouštět, zastavovat a vytvářet nové kontejnery, sítě a Volume. Pro splnění základní podmínky pro schopnost Docker Swarm pokračovat v běhu, je potřeba použít minimálně tři Swarm managery, aby bylo zajištěno většinové kvórum i při výpadku jednoho ze Swarm managerů. Větší počet Swarm manageru nám sice zajistí větší toleranci na jejich výpadky, ale také lehce zvyšuje režii a zátěž na těchto hostech. Dále s počtem Swarm manažerů také roste počet vstupních bodů, přes

které je možné v případě hackerského útoku získat kompletní kontrolu nad celým clusterem. Proto se v praxi nejčastěji používají právě tři Swarm manageři.

Pro můj cluster jsem zvolil strategii, že všechny jeho jednotlivé hosty jsou zároveň také Swarm manageři. Výše uvedené nevýhody v tomto případě silně převažuje možnost tolerance na výpadek až 2 hostů. Což bude pro účely případné prezentace podstatně důležitější.

8 Použité kontejnery

Prezentaci schopností kontejnerové virtualizace pomocí Docker Swarm jsem se rozhodl provést pomocí aplikace jednoduché webové prezentace. Prezentace bude využívat několik kontejnerů pro samotný webový frontend a také konejner pro databázi. Dále bude využit kontejner aplikace swarm-visualizer pro přehledné zobrazení aktuálně běžících kontejnerů na jednotlivých Docker hostech. Pro možnost případného prezentováni i kompletního životního cyklu Dockerové aplikace budu všechny obrazy lokálně vytvářet z Dockerfile získaných z oficiálních GitHub stránek jednotlivých projektů.

Pro účely prezentace jsem se rozhodl pro využití oblíbeného redakčního systému Wordpress jako webového frontendu. Wordpress je open source redakční systém, který je napsán v PHP a jako svůj databázový backend primárně využívá databázový sytém MySQL. Podle statistického šeření provedeného sdružením W3Techs, využívá Wordpress dokonce 30 % veřejně dostupných webových stránek.

Databázový systém je problematická část celého řešení. Protože už od samotného základního návrhu není většina databázových systémů navrhována pro běh v kontejnerech a obzvlášť pro recyklovatelnost. Databázové systémy jsou navrhovány pro co nejstabilnější běh a je možné u některých z nich se setkat dokonce s doporučením je vůbec neprovozovat ve virtualizovaném prostředí. V mém řešení tedy využívám pouze jeden kontejner databázového systému MySQL, který má připojený replikovaný volume napříč clusterem. Nejedná se bohužel o řešení zajišťující vysokou dostupnost. Ale v případě zastavení kontejneru nebo výpadku celého hosta je automaticky spuštěna nová instance, která má přístup k původním datům.

Pro lepší možnost prezentování využívám kontejner s aplikací swarm-visualizer odvozený od obrazu vytvořeného Francisco Mirandaou pro Docker Con 2016. Tato aplikace umožňuje pomocí manažerského Docker hostu získat data o kontejnerech běžících na ostatních hostech a zobrazit je v přehledném formátu na webové stránce.

Protože jsem se rozhodl pro využití a kompilaci vlastních Docker obrazů bude potřeba využít i vlastního registru pro distribuci obrazů napříč clusterem. Regisrty stejně jako v případě databáze bude i registr využívat replikované úložiště napříč clusterem.

8.1 Compose File

Pomocí Compose File je definována celá aplikace, která je určena k prezentaci. Jsou zde popsané jak samotné kontejnery a jejich nastavení, tak jsou zde definovány i sítě a úložiště které budou kontejnery využívat.

Bylo by možné veškeré potřebné části definovat pouze v jednom Compose File. Ale pro možnost jednodušší prezentace a možnosti startovat a zastavovat jednotlivé části zvlášť jsem se rozhodl pro rozdělení definice do několika Compose File.

První Compose File definuje chování kontejneru s programem swarm-visualizer.

```
version: "3.2"
services:
  swarm-visualizer:
    image: localhost:5000/visualizer:latest
    ports:
      - "8080:8080"
    volumes:
      - type: bind
        source: /var/run/docker.sock
        target: /var/run/docker.sock
    deploy:
      mode: global
      placement:
        constraints: [node.role == manager]
```

Je zde definována služba swarm-visualizer, používající stejnojmenný obraz. Směrem ze Swarmu je vystaven port 8080, který je stejný jako port, který mají vystavené samotné kontejnery. Pro schopnost aplikace vizualizovat data ohledně stavu Docker Swarm mají kontejnery připojen sock soubor Dockeru z /var/run/docker.sock na stejnou cestu uvnitř kontejneru. Nasazení kontejnerů je nastaveno, aby byl nasazen právě jeden kontejner na jednoho Docker hosta. Nasazení je pro jistou ještě omezeno, aby se kontejnery nasazovali pouze na hosty, které jsou zároveň Swarm manageři.

```
version: "3.2"
services:
  registry:
    image: budry/registry-arm:latest
    ports:
      - "5000:5000"
    volumes:
      - registry-data-volume:/var/lib/registry
    deploy:
      mode: global
volumes:
  registry-data-volume:
```

Druhý poměrně krátký Compose File definuje službu registru obrazů. Tato služba jako jediná využívá obraz z externího registry. Využívám zde obraz Docker pull budry/registry-arm, jedná se o obraz překompilovaný pro architekturu arm, vyházející z oficiálního Dockerfile registru. V rámci tohoto Compose File je mimo služby také definován volume, který bude registy využívat pro persistenci svých dat. Tento volume, registry-data-volume, je v rámci kontejnerů připojen do složky /var/lib/registry, která je výchozí pro ukládaní obrazů. Stejně jako swarm-visualizer je i služba registry nasazována v globálním módu, tedy právě jeden kontejner na jednoho hosta.

Poslední Compose File definuje tu nejpodstatnější čast a to samotnou webovou prezentaci.

```
version: "3.2"
services:
   wordpress:
     image: localhost:5000/wordpress:latest
     ports:
       - "80:80"
     environment:
       WORDPRESS_DB_HOST: localhost:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
     deploy:
       mode: replicated
       replicas: 3
```

Poslední částí celé aplikace je databázový server, který je ale bohužel kvůli chybě v Docker Stack při práci s ARM architekturou potřeba spouštět mimo Compose File a to přímo z příkazové řádky následujícím příkazem

```
docker service create --name db --replicas=1 --publish 3306:3306 --mout
database-data-volume:/var/lib/mysql --env MYSQL_ROOT_PASSWORD=somewordpress
MYSQL_DATABASE=wordpress MYSQL_USER=wordpress MYSQL_PASSWORD=wordpress
web_database localhost:5000/rpi-mysql
```

9 Zhodnocení přínosu využití platformy Docker

Během práce s clusterem se projevily všechny výhody využití kontejnerové virtualizace, které jsou rozebírány v kapitole o kontejnerové virtualizaci. Ovšem bylo až zarážející, do jaké míry se projevilo zlepšení v rychlosti nasazení aplikace. Během práce na clusteru jsem mimo jiné provedl pokus srovnávajíce rychlost instalaci a nasazení aplikace Apache2 s výchozím nastavením. V obou případech jsem použil předem získané lokální zdroje dat, abych omezil vliv rychlosti připojení na výsledek testu. Závěrem testu bylo že rychlost nasazení s využitím platformy Docker a virtualizace zhruba 30x rychlejší než při standardní instalaci.

Také jsem ale narazil na dosud ne úplně překonané nedostatky přístupu platformy Docker při kombinaci s aplikacemi, které nemají možnost jeho využití v základu svého logického návrhu. Konkrétně tento problém nastává při provozu relačních databázových systému na platformě Docker, kdy v extrémních případech může také dojít ke vzniku nekonzistence až ztrátě dat databáze. V praxi bývá tento problém často řešen buďto vyčleněním databázových systémů z Dockeru, nebo úpravou aplikace, aby pokud možno využívala alternativní přístup jako je například objektové úložiště.

10 Závěr

Tato práce prezentuje možnosti a vlastnosti kontejnerové virtualizace a jejího využití spolu s platformou Docker.

V teoretické části jsou definovány základní pojmy a myšlenky. Je zde uvedena stručná historie jak kontejnerové virtualizace, tak plné virtualizace jakožto původní myšlenky, kterou kontejnerová virtualizace dále rozšiřuje a přetváří. V rámci této části práce je také zmíněn trend nové myšlenky správy systému v odborném slangu označovaný jako "Mazlíček nebo stádo".

U samotné platformy Docker se poté práce věnuje jejím jednotlivým částem. Také je zde probrán celý životní cyklus kontejneru. Neboť přístup Dockerové platformy k tomuto problému je velice inovativní a do jisté míry započal revoluci jak ve správě systémů, tak ve vývoji aplikací.

Praktická část má poté za cíl sestavení samotného clusteru, který je složen z pěti mikropočítačů Raspberry Pi Zero a nezbytného příslušenství pro možnost co nejjednodušší přenosnosti celého clusteru. Na tomto clusteru je na operačním systému Raspbian nainstalován Docker a všech pět mikropočítačů je zapojeno pomocí Docker Swarm do jednoho clusteru. V kontejnerové virtualizaci je na clusteru poté spuštěna webová prezentace skládající se z Wordpressového blogu a databázového MySQL serveru. Na které je možné prezentovat výhody platformy Docker jako je rychlé nasazeni, balancování výkonu, reakce na výpadky kontejnerů a hostů a recyklace kontejnerů. Jako pomůcka k případné prezentaci je na clusteru také nainstalován swarm-visualiser, aby potencionální publikum mělo lepší povědomí o aktuálním stavu clusteru a na jakém z hostů právě běží které kontejnery.

Po zkušenostech s prací s tímto clusterem bych na závěr jen chtěl podotknout, že využití mikropočítačů Raspberry Pi s procesory architektury ARM s sebou přineslo daleko více problémů, než se v počátku práce zdálo. Narazil jsem na mnoho problémů s kompatibilitou jednotlivých obrazů a na chybu při použití kombinace Docker Stack a obrazu MySQL databáze. Případným zájemcům o replikaci podobného clusterového řešení bych i přes několikanásobně vyšší náklady doporučil využít spíše počítačů malého formátu s procesory architektury x86, jako jsou například mini počítače z rodiny Intel NUC.

Seznam citací

[c1] "What is virtualization?" [online] [cit 20.11.2017] Dostupné z: https://www.redhat.com/en/topics/virtualization/what-is-virtualization

[c2] "Intel® ARK" [online] [cit 20.11.2017] Dostupné z: https://ark.intel.com/Search/FeatureFilter?productType=processors&VTD=false&ExtendedPageTables=false&VProTechnology=false

[c3] "Project MAC" [online] [cit 10.02.2018] Dostupné z: http://multicians.org/project-mac.html

[c4] "VMware Continues as #1 Vendor in Worldwide Data Center Automation" [online] [cit 10.02.2018] Dostupné z: https://blogs.vmware.com/management/2017/08/vmware-continues-1-vendor-worldwide-data-center-automation.html

[c5] "ChangeLog-2.6.20" [online] [cit 10.02.2018] Dostupné z: https://mirrors.edge.kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.20

[c6] "innotek GmbH" [online] [cit 10.02.2018] Dostupné z: https://www.virtualbox.org/wiki/innotek

[c7] "chroot(2) - Linux manual page - man7.org" [online] [cit 10.02.2018] Dostupné z: http://man7.org/linux/man-pages/man2/chroot.2.html

[c8] "Chroot prostředí - I - Abclinuxu." [online] [cit 10.02.2018] Dostupné z: http://www.abclinuxu.cz/clanky/bezpecnost/chroot-prostredi-i

[c9] "Jails – High value Virtualization — PHKs Bikeshed." [online] [cit 10.02.2018] Dostupné z: http://phk.freebsd.dk/sagas/jails.html

[c10] "Jails - FreeBSD Wiki." [online] [cit 10.02.2018] Dostupné z: https://wiki.freebsd.org/Jails

[c11] "News - Linux-VServer." [online] [cit 10.02.2018] Dostupné z: http://linux-vserver.org/News

[c11] "Paper - Linux-VServer." [online] [cit 10.02.2018] Dostupné z: http://linux-vserver.org/Paper

[c12] "Linux Containers - LXC - Introduction." [online] [cit 10.02.2018] Dostupné z: https://linuxcontainers.org/lxc/introduction/

[c13] "Linux Container - Proxmox VE." [online] [cit 10.02.2018] Dostupné z: https://pve.proxmox.com/wiki/Linux_Container

[c14] "Docker Engine release notes | Docker Documentation." [online] [cit 10.02.2018] Dostupné z: https://docs.docker.com/release-notes/docker-engine

[c15] "Approved Reseller programme launch PLUS more Pi ... - Raspberry Pi." [online] [cit 10.11.2018] Dostupné z: https://www.raspberrypi.org/blog/approved-reseller

[c16] "Akasa Thermal Solution." [online] [cit 10.11.2017] Dostupné z: http://www.akasa.com.tw/search.php?seed=AK-HB-09BK

[c17] "How much power does Pi Zero W use? – RasPi.TV." [online] [cit 10.11.2017] Dostupné z: http://raspi.tv/2017/how-much-power-does-pi-zero-w-use.

[c18] "Mechanical Drawings - Raspberry Pi Documentation" [online] [cit 20.12.2017] Dostupné z: https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical/README.md

[c19] "RaspbianFAQ - Raspbian." [online] [cit 05.01.2018] Dostupné z: https://www.raspbian.org/RaspbianFAQ

[c21] "Download Raspbian for Raspberry Pi" [online] [cit 05.01.2018] Dostupné z: https://www.raspberrypi.org/downloads/raspbian/

[c22] "Download Raspbian for Raspberry Pi" [online] [cit 05.01.2018] Dostupné z: https://downloads.raspberrypi.org/raspbian_lite_latest

[c23] "Ubuntu Manpage: sha256sum - compute and check SHA256" [online] [cit 05.01.2018] Dostupné z: http://manpages.ubuntu.com/manpages/trusty/man1/sha256sum.1.html

[c24] "Ubuntu Manpage: dd - convert and copy a file." [online] [cit 05.01.2018] Dostupné z: http://manpages.ubuntu.com/manpages/trusty/man1/dd.1.html

Seznam zdrojů

[z1] Vlastní archív

[z2] SEEKLOGO.COM. Docker logo [online]. [cit. 14.3.2018]. Dostupné z: https://seeklogo.com/images/D/docker-logo-6D6F987702-seeklogo.com.png

Seznam použité literatury

MATTHIAS, Karl a Sean P. KANE. Docker: Up & Running: Shipping Reliable Containers in Producton. O'Reilly Media. ISBN 978-1491917572.

SOPPELSA, Fabrizio a Chanwit KAEWKASI. Natve Docker Clustering with Swarm. Packt Publishing 2016.ISBN 9781786469755.

TURNBULL, James. The Docker Book. 2014. ISBN 978-0-9888202-0-3.

UPTON, Eben; HALFACREE, Gareth. Raspberry Pi-uživatelská příručka. Computer Press, Albatros Media as, 2017.

NORRIS D. Raspberry Pi: Projekty. Computer Press, 2015. ISBN 978-80-251-4346-9