Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Big data a kontejnerová virtualizace

Bakalářská práce

Autor: David Vondráček
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Michal Macinka

Hradec Králové červen 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Přelouči dne 14.08.2020

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Michalu Macinkovi za metodické vedení práce.

Anotace

Cílem této bakalářské práce je uvést do problematiky zpracování velkých dat za pomoci kontejnerizačních platforem Docker a Kubernetes. Bakalářská práce je rozdělená do čtyř částí. První část se věnuje velkým datům, jejich historii, metodám jejich analýzy a nástrojům, pomocí kterých lze velká data zpracovávat. Druhá část se zabývá kontejnerovou virtualizací se zaměřením na kontejnerové platformy Docker a Kubernetes, u kterých je vysvětlena klastrová architektura. Součástí kapitoly je také popsání kontejnerizace, její výhody, nevýhody a také porovnání kontejnerů a virtuálních strojů. Třetí část vysvětluje využití velkých dat prostřednictvím otevřeného software Docker. Poslední část je zaměřena na implementaci skriptu pro UNIX.

Annotation

Title: Big data and Container Virtualization

The aim of this bachelor thesis is to introduce the problematics of big data processing using containerization platforms Docker and Kubernetes. The bachelor thesis is divided into four parts. The first part deals with big data, their history, methods of their analysis and tools that can be used to process big data. The second part deals with the container virtualization with a focus on the Docker and Kubernetes container platforms, for which the cluster architecture is explained. The chapter also describes the containerization, its advantages, disadvantages and also a comparison of containers and virtual machines. The third part explains the use of big data through open Docker software. The last part is focused on the implementation of the script for UNIX.

Obsah

1 Úvod

Termín „big data“ neboli velká data (VD) lze vysvětlit mnoha odlišnými způsoby, které na jejich podstatu nahlížejí z různých úhlů pohledu. Tento termín lze popsat jako nový přístup ke zpracování velkého množství dat. Tato data vznikají vysokým tempem, jejich podoba není pevně daná a klasické přístupy pro zpracování těchto dat jsou nedostačující. Některé aspekty tohoto pojmu tato výše popsaná definice dobře vystihuje. Pro lepší pochopení problematiky VD je však nutné tento pojem popsat v širších souvislostech. Jedná se o aktuální problematiku, a její vývoj je v poslední době jedním z nejrychlejších v oblasti informačních technologií, možná i proto pro pojem VD najdeme v literatuře mnoho různých vysvětlení [1] [2].

Ke skutečnému porozumění VD, je vhodné znát historické pozadí. Kolem roku 2001 přišla firma Gartner s definicí: „*Velká data jsou data, která obsahují větší rozmanitost, přicházející ve zvyšujících se objemech a se stále vyšší rychlostí*“ [3]. Tato definice je také známá jako tři V [3].

VD jsou větší a složitější sady dat, zejména z nových zdrojů dat. Tyto datové sady jsou tak objemné, že tradiční software pro zpracování dat je nemůže spravovat. Tyto obrovské objemy dat však lze použít k řešení obchodních problémů, s kterými se dříve nešlo vypořádat [3].

Tři V o VD [3]:

1. Volume (objem): Záleží na množství dat. U VD je třeba zpracovávat velké objemy nestrukturovaných dat s nízkou hustotou. Tyto data mohou být neznámých hodnot, jako jsou datové zdroje Twitter, nebo mobilní aplikace. Pro některé organizace to může být desítky terabytů, pro jiné organizace zase stovky petabytů dat.
2. Velocity (rychlost): Rychlost, při které jsou data přijímána, obvykle je nejvyšší rychlost datových toků přímo do paměti, oproti zápisu na disk. Některé internetové produkty fungují v reálném čase a vyžadují vyhodnocení a akci v reálném čase.
3. Variety (rozmanitost): Týká se mnoha typů dat, která jsou k dispozici. Tradiční datové typy byly strukturovány a zapadaly do relačních databází. Se vzestupem

VD přicházejí data do nových nestrukturovaných datových typů. Nestrukturované datové typy, jako je text, zvuk a video, vyžadují další předzpracování, aby bylo možné odvodit jejich význam.

V posledních několika letech se objevily další dvě V: hodnota (value) a pravdivost (veracity). Data mají vlastní hodnotu, dokud se však tato hodnota neobjeví, tak jsou data k ničemu. Stejně důležitá je pravdivost zkoumaných dat a jejich spolehlivost. Dnes se VD stala kapitálem. Velká část hodnoty, kterou nabízejí největší světové technologické společnosti, pochází z dat, která neustále analyzují, aby dosáhly vyšší efektivity a vývoje nových produktů. Nedávné technologické objevy exponenciálně snížily náklady na ukládání a výpočet dat. Se zvýšeným objemem VD, které jsou nyní levnější a dostupnější, lze činit přesnější obchodní rozhodnutí. Nalezení hodnoty ve VD není jen o jejich analýze, jedná se o celý proces zkoumání vyžadující analytiky a management, který si klade správné otázky, rozpoznává vzorce a předpovídá chování dat [3].

2 Cíl práce

Cílem této bakalářské práce bude popsat problematiku velkých dat, jejich velikost a způsoby uložení. V práci se představí metody analýzy velkých objemů dat, pro zpracování velkých dat budou vysvětleny nástroje jako jsou Hadoop, Spark a Storm. Práce bude popisovat, co je to kontejnerová virtualizace, její výhody, nevýhody a k čemu je dobré jí využívat. Dále se zaměří na problematikou kontejnerů a kontejnerových platforem, konkrétně Docker a Kubernetes. U Dockeru bude vysvětleno, pro koho je vhodný, co je Docker Swarm a Docker Compose, kapitola s Docker Swarmem seznámí s principem uzlů a klastrů. Kubernetes popíše dobu nasazení aplikací a klastrovou architekturu. V poslední část teorie se uvede využití kontejnerové virtualizace pro analýzu velkých dat.

V praktické části se navrhne a realizuje možné ukázkové řešení problematiky analýzy velkých dat s využitím kontejnerové virtualizace. Řešení bude realizováno formou skriptu pro UNIX s využitím kontejnerových platforem Docker a Kubernetes, nasazením Hadoopu do Dockeru, použití Docker Swarmu a klastru v Kubernetes. Úvodní část seznámí s principem skriptu a jeho funkcí. Další část vysvětlí nasazení Docker Swarmu jak do samotného operačního systému, tak do Docker Machine. V obou případech bude vysvětleno, jak spustit požadovaný klastr a co se stane, když vypadne server. V poslední části bude popsáno inicializace klastru v Kubernetes.

3 Big data a analýza velkých objemů dat

Přestože je koncept VD sám o sobě relativně nový, počátky velkých datových souborů sahají až do šedesátých a sedmdesátých let, kdy první datová centra začala s vývojem relačních databází. Kolem roku 2005 si lidé začali uvědomovat, kolik uživatelé vygenerovali dat prostřednictvím služeb Facebook, YouTube a dalších online služeb. Ve stejné roce byl vyvinut Hadoop a NoSQL během této doby začalo získávat na popularitě.
Vývoj otevřených frameworků jako je Hadoop, byl nezbytný pro růst VD, protože usnadňuje práci s daty a snižuje cenu ukládání. Od té doby se objem VD prudce zvýšil. S příchodem Internet of Things (IoT) je k internetu připojeno více objektů a zařízení, jež shromažďují data. Vznik strojového učení přinesl ještě více dat a přestože VD mají dnes veliké využití, jejich plný potenciál ještě nebyl plně dosáhnut [3].

3.1 Big data a jejich velikost

VD jsou ve velikosti petabytů (PB), exabytů (EB) nebo zettabytů (ZB). Jeden PB odpovídá 20 miliónům tradičních kartoték textů [4]. VD jsou o rychlosti tvorby dat. Mnoho zdrojů VD poskytuje různorodé bohatství, které zdaleka překonává tradiční data z minulosti. Hlavním rozdílem mezi současnými VD a tradičními daty je přechod ze strukturovaných transakčních dat na nestrukturovaná data [4].
Za posledních 20 let se data zvýšila ve velkém rozsahu v různých oblastech. V roce 2011 podle Mezinárodní Datová Korporace (IDC) byl celkový vytvořený a zkopírovaný objem dat na světě 1,8 ZB, což je devětkrát více než před 5 lety. Taková hodnota se zdvojnásobí nejméně každé další 2 roky v blízké budoucnosti [5]. Termín VD byl vytvořen díky prudkému nárustu globálních dat a byl používán hlavně k popisu těchto enormních datových souborů. Ve srovnání s tradičními datovými soubory, VD obecně zahrnují množství nestrukturovaných dat, která potřebují více analýzy v reálném čase. VD navíc přinášejí nové příležitosti objevování jejich hodnot, pomáhají nám získat hloubkové porozumění a přinášejí nové výzvy, například o tom, jak efektivně organizovat a spravovat taková data. V současné době VD vzbudily velký zájem ze strany průmyslu, akademických a vládních agentur [5].
Rychlý růst VD pochází především z každodenního života lidí, související zejména se službami internetových firem. Například Google zpracovává data stovek PB a Facebook

generuje log dat přes 10 PB [5]. Množství velkých datových souborů drasticky roste a přináší mnoho náročných problémů, jak tyto data rychle a efektivně zpracovávat. Nejnovější IT technologie umožňují snadnější generovaní dat. V průměru je každou minutu nahráno 72 hodin videí na YouTube [6]. To vytváří výzvu pro sbírání a integraci masivních dat, ze široce distribuovaných zdrojů dat. Množství shromážděných dat stále více roste, což způsobuje problém, jak je ukládat a spravovat. S ohledem na různorodost, škálovatelnost, reálný čas a složitost VD, by bylo vhodné účinně „těžit" soubory dat na různých úrovních s analýzou, modelováním, vizualizací a optimalizací techniky, aby bylo možné odhalit jejich vnitřní vlastnosti a zlepšit rozhodování co s těmito daty dělat. Rychlý růst cloud computingu a IoT podporuje prudký nárůst dat. Cloud computing poskytuje ochranu, přístup a kanály pro data. V IoT sbírají senzory po celém světe data, která budou uložena a zpracována v cloudu. Tato data výrazně převyšují kapacity IT architektur a infrastruktur stávajících podniků, a jejich požadavky značně rostou [5].

3.2 Big data a jejich úložiště

Na základě analýzy technologií pro ukládaní dat bylo získáno několik poznatků. Ukládání VD se stalo komoditou a škálovatelné technologie pro ukládání dat dosáhly podnikové úrovně, která dokáže spravovat prakticky neomezené objemy dat. Důkazem je rozšířené používání Hadoopu nabízené dodavateli, jako je Cloudera, Hortonworks a MapR, stejně jako databáze NoSQL, ve srovnání s tradičními systémy správy relačních databází, které se spoléhá na úložiště a drahé ukládání do mezipaměti. Databázové technologie nabízejí lepší škálovatelnost a nižší náklady, přes tyto pokroky zlepšující výkon, škálovatelnost a použitelnost je stále ještě nevyužitý potenciál ukládání VD. Existuje stále mnoho problémů ukládaní, použití a dalších rozvojů VD [7].

Potenciální transformace společnosti a podniků napříč odvětvími: technologie ukládání VD jsou klíčovým nástrojem pro pokročilé analýzy, které mají transformovat společnost a způsob, jakým jsou přijímána klíčová obchodní rozhodnutí. To je obzvláště důležité v odvětvích, které nejsou tradičně založena na IT, jako jsou energie. Největším problémem tohoto odvětví je nedostatek expertů pro VD. Nové technologie ukládání dat mají umožnit nové analytické metody v různých průmyslových odvětvích [7].

Nedostatek standardů je hlavní bariéra: Historie NoSQL je založena na řešení specifických technologických výzev, které vedou k řadě různých metod ukládání. Velký

výběr možností spojených s nedostatkem standardů pro dotazování dat ztěžují výměnu datových úložišť [7].

Otevřené výzvy škálovatelnosti v grafických úložištích dat: Zpracování dat založené na grafových strukturách je přínosné ve stále větším počtu aplikací. Umožňuje lepší zachycení sémantiky a složitých vztahů s dalšími informacemi pocházejících z různých zdrojů dat a má potenciál zlepšit celkovou hodnotu, kterou lze vytvořit analýzou dat. Zatímco k tomuto účelu jsou stále více využívány databáze grafů, je složité efektivně distribuovat jejich datovou strukturu. Ochrana osobních údajů a bezpečnost není dostačující, i když existují projekty a nová řešení, která se zabývají problematikou bezpečnosti a ochrany osobních údajů. Ochrana jednotlivců a osob stále zaostává za technologickým pokrokem při ukládání dat do systémů [7].

3.3 Analýza velkých objemů dat

Analýza velkých objemů dat vyžaduje použití vhodných statistických metod, které je třeba seskupit, extrahovat a vybrat z nich užitečná data. Analýza dat hraje velkou roli při tvorbě rozvojových plánů a pochopení požadavků zákazníků a předvídání trhu podniků. Analýzu VD lze považovat za analýzu speciálního druhu dat. Proto může být pro VD stále využíváno mnoho tradičních metod analýzy dat [5].

Na počátku období VD se lidé zajímali o rychlé extrahování klíčových informací z masivních dat. Přinášely hodnoty pro podniky a pro jednotlivce. V současné době jsou hlavní metody zpracování VD následující [5]:

- **Klastrová analýza**: je statistická metoda pro seskupování objektů a jejich třídění podle vlastností. Klastrová analýza se používá k rozlišení objektů s určitými vlastnostmi a jejich rozdělení do kategorií (klastrů) podle těchto vlastností, například objekty ve stejné kategorii budou mít vysokou homogenitu, v různých kategoriích budou mít vysokou heterogenitu.
- **Faktorová analýza**: je zaměřena na popis vztahu mezi mnoha ukazateli nebo prvky s několika málo faktory, to je seskupením několika úzce souvisejících proměnných. Každá skupina proměnných se poté stává faktorem. Poté je několik faktorů použito k odhalení nejcennějších informacích o originálních datech.

- **Korelační analýza**: je analytická metoda mezi pozorovanými jevy a podle toho jsou prováděny prognózy a kontroly. Existuje spousta kvantitativních vztahů mezi pozorovanými jevy jako korelace, korelační závislost a vzájemného omezení. Tyto vztahy lze rozdělit do dvou typů. Prvním typem je funkce, která odráží přísný vztah závislosti mezi jevy, kde každá číselná hodnota nebo proměnná odpovídá jedné nebo více určených hodnot. Druhým typem je korelace, kde existují neurčené a nepřesné vztahy závislostí a číselná hodnota může odpovídat několika číselným hodnotám jiné proměnné.
- **Regresní analýza**: je matematický nástroj pro odhalení korelace mezi jednou proměnnou a několika dalšími proměnnými. Na základě pozorovaných dat regresní analýza identifikuje závislost mezi proměnnými. Regresní analýza může změnit složité a neurčené korelace mezi proměnnými na jednoduché a pravidelné korelace.
- **Statistická analýza**: je založena na statistické teorii. Ve statistické teorii je náhodnost modelována s teorií pravděpodobnosti. Statická analýza může poskytnout popis pro rozsáhlé sobory dat.
- **Dolování dat (Data mining)**: je proces získávání skrytých, neznámých, ale potenciálně užitečných informacích a znalostí z masivních, neúplných a náhodných dat. Dolování dat se používá především k dokončení různých úkolů: klasifikace, odhad, predikce. Originální data mohou být strukturována, například data v relačních databázích, polostrukturovaná data, textová, grafická nebo obrazová data. Zjištěné znalosti mohou být použity pro správu informací, optimalizaci dotazů, podporu rozhodování a proces kontroly, jakož i údržby dat[5].
- **Bloom Filter**: je bitové pole a řada hash funkcí. Princip bloom filtru je ukládat hash hodnoty jiných dat, než jsou data bitového pole, což je v podstatě bitmapový index, který používá hash funkce k ukládání ztrátových kompresních dat. Má takové výhody jako vysoká efektivita prostoru a vysoká rychlost dotazování, ale také některé nevýhody, jako je míra chybného rozpoznání, nebo obtíže s vymazáním.
- **Hashování**: je metoda, která transformuje data na kratší délku číselné hodnoty, nebo hodnoty indexu. Hashování má výhody v rychlém čtení, zapisování a vysokou rychlostí dotazování, ale je složitější na nalezení.

- **Index**: je vždy efektivní metodou ke snížení nároků zapisování a čtení na disku. Zlepšuje vkládání, mazání, úpravy a rychlost dotazování v tradičních a relačních databázích. Index má nevýhodu, že má dodatečné nároky na ukládání indexových souborů a soubory indexu by měly být dynamické, vzhledem k aktualizaci dat.
- **Triel**: také nazývaný jako strom je varianta hashovacího stromu, uplatňuje se především na vyhledávání četnosti slov. Hlavní myšlenkou trielu je využití společné předpony řetězců znaků ke snížení porovnávání, aby se zvýšila efektivita dotazování.

3.4 Nástroje pro zpracování velkých dat

Apache Hadoop

Obrázek 1 Logo Apache Hadoopu. Zdroj: [8]

Apache Hadoop je framework, který umožňuje distribuované zpracování velkých datových souborů napříč klastry počítačů, pomocí jednoduchých programovacích modelů. Je navržen tak, aby se rozšířil od jednotlivých serverů po tisících počítačů, z nichž každý nabízí místní výpočetní výkon a úložiště. Spíše než se spoléhat na hardware, který poskytuje vysokou dostupnost, je framework sám navržen tak, aby detekoval a řešil selhání v aplikační vrstvě. Díky tomu poskytuje vysoce dostupnou službu na vrcholu klastru počítačů, z nichž každý může být náchylný k selhání [9].
Výhody použití Apache Hadoopu jsou například [9]:

- vylepšení ověřování při použití HTTP proxy serveru,
- specifikace pro kompatibilní souborový systém Hadoopu,
- nabízí robustní ekosystém, který je vhodný pro splnění analytických potřeb vývojáře,
- přináší flexibilitu při zpracovávání dat,
- umožňuje rychlejší zpracovávání dat.

Apache Spark

Obrázek 2 Logo Apache Sparku. Zdroj: [10]

Apache Spark je dalším nástrojem pro zpracování VD. Klíčovým bodem tohoto otevřeného software je doplnění nedostatků Apache Hadoopu v problematice zpracování VD. Spark dokáže zpracovávat jak dávková data, tak data v reálném čase, protože Spark zpracovává data v paměti, tak je zpracovává mnohem rychleji než tradiční disky. Pro datové analytiky, kteří zpracovávají určité typy dat, je to plusový bod k dosažení rychlejšího výsledku [11]. Apache Spark je flexibilní pro práci s HDFS (= distribuovaný systém souborů Hadoop), i s jinými datovými úložišti, například s OpenStack Swift, nebo Apache Cassandra. Spark lze také snadno provozovat na jednom místním systému, což usnadňuje vývoj a testování.
Spark Core je jádrem projektu a usnadňuje mnoho věcí, mezi které patří například distribuovaný přenos úloh, plánování a I/O funkčnost [11].

Apache Storm

Obrázek 3 Logo Apache Stormu. Zdroj: [12]

Apache Storm je bezplatný, otevřený software, který zpracovává data v reálném čase. Usnadňuje spolehlivé zpracování neomezených toků dat v reálném čase. Apache Storm je jednoduchý a lze použít s jakýmkoli programovacím jazykem. Apache Storm má mnoho využití: analýza v reálném čase, online strojové učení, nepřetržitý výpočet, distribuované RPC (= vzdálené volání procedur), ETL (= extrakce, transformace, nahrání) a další. Je škálovatelný, odolný vůči chybám (zaručuje, že data budou zpracována), je snadno nastavitelný a provozovatelný. Apache Storm se integruje do frontových a databázových technologií. Jeho topologie spotřebovává datové toky, zpracovává je komplexními způsoby, mezi každou fází výpočtu je rozděluje dle potřeby [13].

Má mnoho výhod a některé z jeho klíčových funkcí jsou [13] [14]:

- uživatelsky přívětivý,
- ideální jak pro malé, tak pro velké implementace,
- vysoce odolný vůči chybám,
- spolehlivý,
- umožňuje zpracování v reálném čase,
- je velice rychlí, protože má obrovskou sílu zpracování dat,
- vysoce škálovatelný,
- poskytuje zaručené zpracování dat, i když dojde ke ztrátě některého z připojených uzlů v klastru.

4 Kontejnerová virtualizace

Kontejnery jsou typ softwaru, který dokáže virtuálně zabalit a izolovat aplikace pro jejich nasazení. Kontejnery mohou sdílet přístup do jádra operačního systému (OS), bez potřeby virtuálních strojů (VM = Virtual Machine). Kontejnerová technologie byla vyvinuta ve formě izolace procesů, která byla součástí Linuxu v sedmdesátých letech [15]. Jeho moderní forma je vyjádřena v aplikační kontejnerizaci, jako je Docker a systémové kontejnerizaci, jako jsou linuxové kontejnery (LXC = Linux Containers). Oba tyto styly kontejnerů zjednodušují správu verzí a umožňují přenositelnost v různých prostředích nasazení [15].

Kontejnerové obrazy obsahují informace, které se provádějí za běhu OS pomocí kontejnerového engine. Kontejnerizované aplikace se mohou skládat z několika obrazů kontejnerů. Například aplikace implementované tří vrstvou architekturou mohou být složeny z front-endového, back-endového a databázových serveru umístěných uvnitř kontejnerů, které se spouští nezávisle [15]. Kontejnery mohou spustit více instancí obrazu a nové instance mohou nahradit poškozené instance, aniž by došlo k narušení provozu celé aplikace. Vývojáři používají kontejnery během vývoje a testování a stále více IT týmů se snaží nasazovat produkční prostředí na kontejnery především v cloudu [15].

Výhody použití kontejnerizace

Kontejnerizace dosáhla popularity s otevřeným software Docker. Docker vyvinul techniku, která poskytuje kontejnerům vetší flexibilitu a umožňuje přesouvat softwarovou aplikaci mezi všemi systémy. Kontejnerizace je podpora při dosažení maximální efektivity pro ukládání, CPU a paměti, to je významná výhoda, která je dosažena kontejnerizací místo tradiční virtualizace, protože kontejnery nemají samostatný OS a jednotnou infrastrukturu, tak může podporovat více než jeden kontejner. Aplikační kontejnerizace může rozšířit výkon, protože jednoduše umožňuje jedinému OS, aby se postaral o všechny hardwarové volání. Jednou z hlavních výhod kontejnerů je, že mohou být snadno a rychle vyvinuty, což je perfektní řešení pro vývoj agilního prostředí, které usnadňuje různé techniky, jako jsou mikroslužby [16].

Nevýhody použití kontejnerizace

Kromě mnoha výhod má kontejnerová virtualizace také několik nevýhod. Jednou z hlavních nevýhod je nedostatečná izolace z hostitelského OS, kontejnery sdílí hostitelský OS. Existují tedy bezpečností hrozby, které mohou poškodit celý systém. Další nevýhodou kontejnerizace je základní použití stejného OS, protože v hypervisoru každá operace vyžaduje svůj OS [16].

4.1 Porovnání kontejnerů a virtuálních strojů

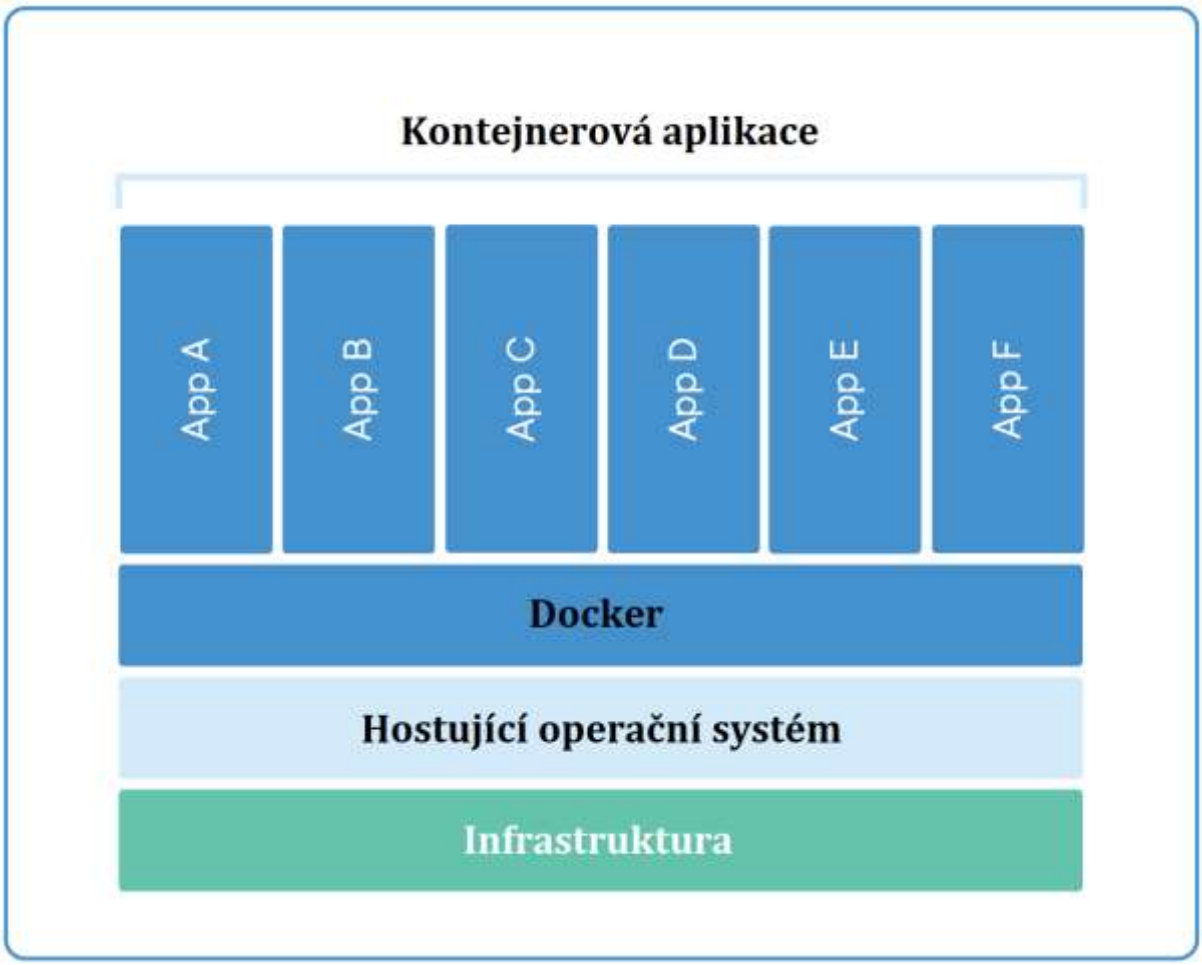
Kontejnery a VM mají podobné výhody izolace a přidělení prostředků, ale fungují odlišně, protože kontejnery virtualizují OS místo hardwaru. Kontejnery jsou více přenosnější a efektivnější [17].

Kontejnery

Kontejnery jsou formou virtualizace OS, ale na rozdíl od VM nevirtualizují hardware, a proto jsou méně náročné na zdroje než VM. Více kontejnerů může běžet na stejném počítači a sdílet kernel OS s jinými kontejnery, z nichž každý běží jako izolovaný proces v uživatelském prostoru. Kontejnery vyžadují méně RAM, CPU a zabírají méně místa než VM (obrazy kontejnerů jsou typicky ve velikosti desítky MB) [18].
Kontejner je standardní jednotka softwaru, která obaluje kód a všechny jeho závislosti díky tomu běží rychle a spolehlivě ve výpočetních prostředích. Docker kontejner je spustitelný balíček softwaru, který obsahuje vše potřebné pro spuštění aplikace: kód, běhové prostředí, systémové nástroje, systémové knihovny a konfigurace. Kontejnerové obrazy jsou šablonou pro konkrétní spuštěné instance kontejnerů. Kontejnerový software, který je k dispozici pro OS Linux, Windows a macOS, bude vždy fungovat stejně, bez ohledu na infrastrukturu. Kontejnery izolují software ve svém prostředí a zajišťují jeho jednotnou funkci [17].
Výhody kontejnerů [19]:

- agilní tvorba a nasazení aplikací: snadnější a efektivnější vytváření obrazů kontejnerů ve srovnání s použitím obrazu VM,

- nepřetržitý vývoj, integrace a rozmístění: zajišťuje spolehlivé a časté vytváření a rozmístění obrazů kontejnerů s rychlým a snadným vrácením (kvůli zaměnitelnosti obrazu),
- vytváření obrazů aplikačních kontejnerů v době sestavení, nikoli v době nasazení, čímž se oddělují aplikace od infrastruktury,
- soulad prostředí mezi vývojem, testováním a výrobou: běží na notebooku stejně jako v cloudu,
- přenositelnost distribuce v cloudu a OS: běží na Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine a kdekoli jinde,
- správa zaměřená na aplikaci: zvyšuje úroveň abstrakce od spuštění OS na virtuálním hardwaru po spuštění aplikace v OS pomocí logických prostředků,
- volně vázané, distribuované, uvolněné mikro-služby: aplikace jsou rozděleny na menší, nezávislé kusy a lze je nasadit a dynamicky řídit, nejedná se o monolitický zásobník běžící na jednom velkém jednoúčelovém stroji,
- izolace zdrojů: předvídatelný výkon aplikace,
- využití zdrojů: vysoká účinnost a hustota.

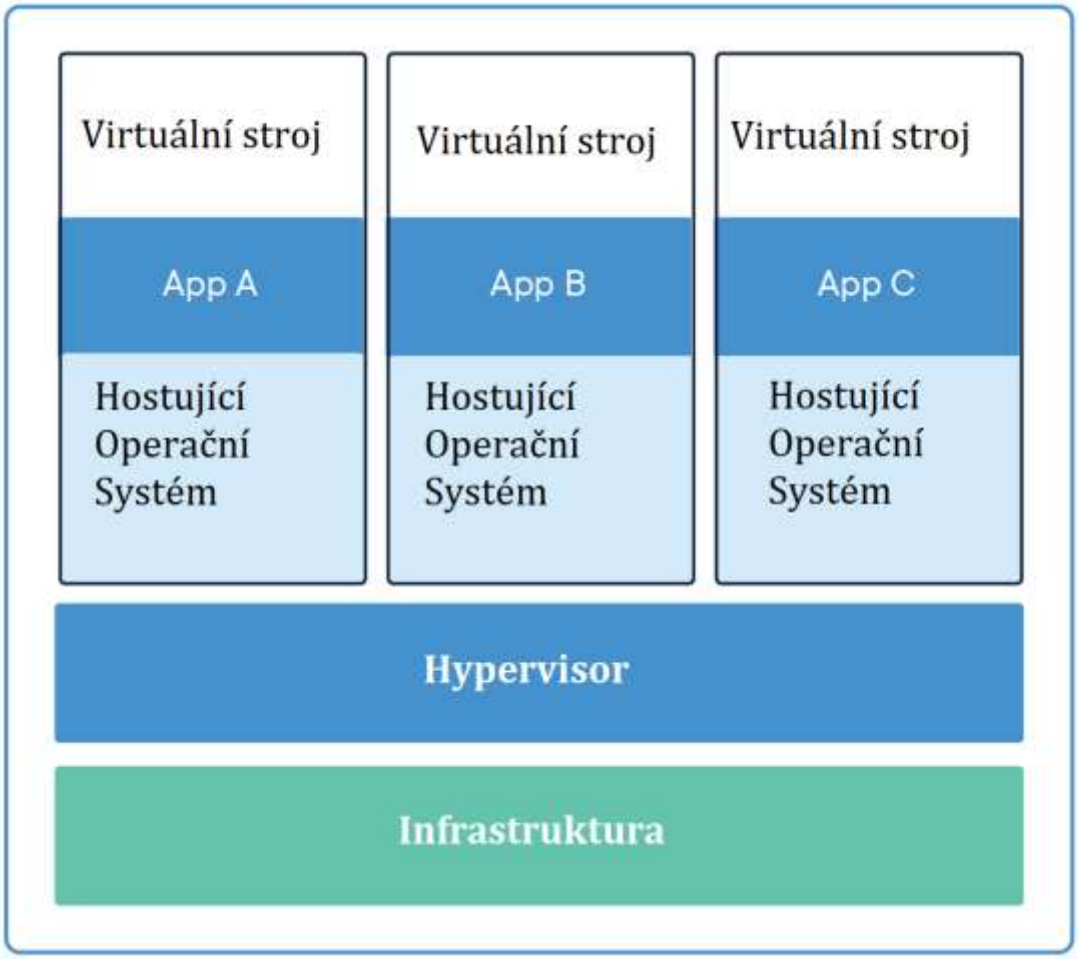


Obrázek 4 Kontejnerové aplikace. Zdroj: [17]

Virtuální stroje

VM je počítačový soubor, který se chová jako skutečný počítač. Jinými slovy, vytvoření počítače v počítači. VM je izolován od systému, což znamená, že software uvnitř VM nemůže manipulovat se samotným počítačem, to vytváří ideální prostředí pro testování dalších OS, přístupu k datům napadeným virem, vytváření záloh OS a spouštění softwaru nebo aplikací v OS, pro který nebyly původně určeny [20].

Na stejném fyzickém počítači může běžet současně více VM pomocí hypervisoru, který je spravuje, každý VM poskytuje svůj vlastní virtuální hardware, včetně procesoru, paměti, pevných disků, síťových rozhraní a dalších zařízení. Virtuální hardware je poté mapován na skutečný hardware na fyzickém stroji, který šetří náklady snížením potřeby fyzických hardwarových systémů spolu s náklady na údržbu, navíc snižuje spotřebu energie a chlazení [20].



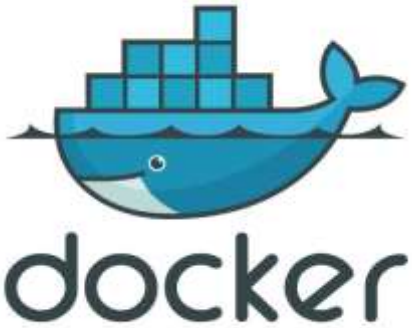
Obrázek 5 Struktura virtuálního stroje. Zdroj: [17]

Hypervisor

Hypervisor je počítačový software nebo hardware, který umožňuje hostovat více VM a každý VM je schopen spouštět své vlastní programy. Hypervisor umožňuje přístup k několika VM, které využívají hardware jednoho počítače. Každý VM obsahuje procesor, paměť a prostředky hostitelského hardware, ale ve skutečnosti je to hypervisor, který přiděluje tyto prostředky VM [21].

První hypervisory byly představeny v 60. letech, aby umožnily souběžný běh různých OS na jednom počítači. Jeho současná popularita je hlavně způsobena UNIXem. Kolem roku 2005 začaly systémy UNIX používat virtualizační technologii k rozšíření hardwarových možností, řízení nákladů, zvýšení spolehlivosti a zabezpečení [21].

4.2 Docker



Obrázek 6 Docker logo. Zdroj: [22]

Kontejnerová technologie Docker byla spuštěna v roce 2013 jako otevřený software Docker Engine. To využilo existující počítačové koncepty kolem kontejnerů a specificky ve světě Linuxu. Technologie Dockeru je unikátní, protože se zaměřuje na požadavky vývojářů a systémových operátorů na oddělení závislosti aplikací od infrastruktury.

Úspěch ve světě Linuxu vedl k partnerství s Microsoftem, který přinesl Docker kontejnery a jejich funkčnost na Windows Server (někdy označované jako Docker Windows kontejnery) [17].

Technologie, která je k dispozici od společnosti Docker a jejího otevřeného projektu, využívá všech hlavních dodavatelů datových center a poskytovatelů cloudových služeb. Mnozí z těchto poskytovatelů využívají Docker pro své nativní kontejnery, kromě toho otevřené frameworky využívají kontejnerovou technologii Docker [17].

Docker je nástroj určený k usnadnění vytváření, nasazení a spouštění aplikací pomocí kontejnerů. Kontejnery umožňují vývojáři zabalit aplikaci se všemi částmi, které potřebuje, jako jsou knihovny a další závislosti a odeslat je jako jeden balíček. Díky kontejneru si vývojář může být jistý, že aplikace bude běžet na jakémkoli OS [23].

Docker je jako VM, ale na rozdíl od VM Docker vytváří celé virtuální OS a umožňuje aplikacím používat stejné jádro, jako systém, na kterém běží. Docker vyžaduje, aby aplikace byly dodávány pouze s programy, které na hostitelském počítači ještě nejsou spuštěny, to významně zvyšuje výkon a snižuje velikost aplikace. Docker je otevřený software, to znamená, že kdokoli může do softwaru přispět a rozšířit jej tak, aby vyhovoval jeho vlastním potřebám [23].

Pro koho je Docker

Docker je navržený tak, aby byl přínosem jak pro vývojáře, tak pro správce systému. Je součástí mnoha nástrojů DevOps (Development + Operations = vývoj + operace). Pro vývojáře to znamená, že se mohou zaměřit na psaní kódu bez obav o systém, na kterém bude nakonec spuštěn. Rovněž umožňuje spuštění pomocí jednoho z tisíců programů, které jsou již navrženy v Docker kontejneru jako součást aplikace. Pro operace poskytuje Docker flexibilitu a potenciálně snižuje počet potřebných systémů kvůli své malé stopě a nižší režii [23].

Docker Swarm

Docker Swarm je skupina fyzických, nebo VM, které používají Docker a které byly nakonfigurovány tak, aby se spojily v klastru. Po seskupení skupiny strojů lze stále spouštět Docker příkazy, ale budou je provádět stroje v klastru. Činnosti klastru jsou řízeny správcem Swarmu a stroje, které se klastru připojily, se označují jako uzly. Docker Swarm je nástrojem orchestrace kontejnerů, což znamená, že uživateli umožňuje spravovat více kontejnerů rozmístěných na více hostitelských strojích. Jednou z klíčových výhod spojených s Docker Swarmem je vysoká úroveň dostupnosti nabízené aplikacím. V Docker Swarmu je obvykle několik pracovních uzlů a alespoň jeden správcovský uzel, který je zodpovědný za efektivní zacházení se zdroji pracovních uzlů a za zajištění fungování klastru [24].

Uzly v Docker Swarmu

Docker Swarm se skládá ze skupiny fyzických nebo VM pracující v klastru. Když se stroje připojí ke klastru, stanou se uzlem v tomto klastru. Technologie Docker Swarm má tři různé typy uzlů, z nichž každý má jinou roli [24]:

1. **Uzel správce**: Primární funkcí uzlů správce je přiřazovat úkoly pracovním uzlům ve Swarmu. Uzly správce také pomáhají provádět některé z manažerských úkolů potřebných k ovládání Swarmu. Docker doporučuje pro Swarm maximálně sedm manažerských uzlů [24].
2. **Pracovní uzel**: Pracovní uzly fungují tak, že přijímají a provádí úkoly, které jim jsou přiděleny uzly správce. Ve výchozím nastavení jsou všechny uzly správce také pracovními uzly a jsou schopny provádět úkoly, když mají k dispozici prostředky.

Docker Compose

Compose je nástroj pro definování a spuštění více kontejnerů Docker aplikací. Pomocí Composu lze využít soubor YAML ke konfiguraci služeb aplikace. Jedním příkazem se spustí všechny služby z konfigurace. Compose pracuje ve všech prostředích: pracovní, produkční, vývojové, testovací [25].

Použití Docker Composu je tříkrokový proces, po kterém se spustí celá aplikace běžící na kontejnerech [25]:

1. definice prostředí aplikace pomocí Dockerfile, aby bylo možné kdekoliv reprodukovat,
2. definování služeb v `docker-compose.yml`, které tvoří aplikaci, aby mohly být provozovány společně v izolovaném prostředí,
3. spuštění `docker-compose up` a Compose spustí celou aplikaci.

`docker-compose.yml` může vypadat následovně:

```
version: '2.0'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
logvolume01: {}
```

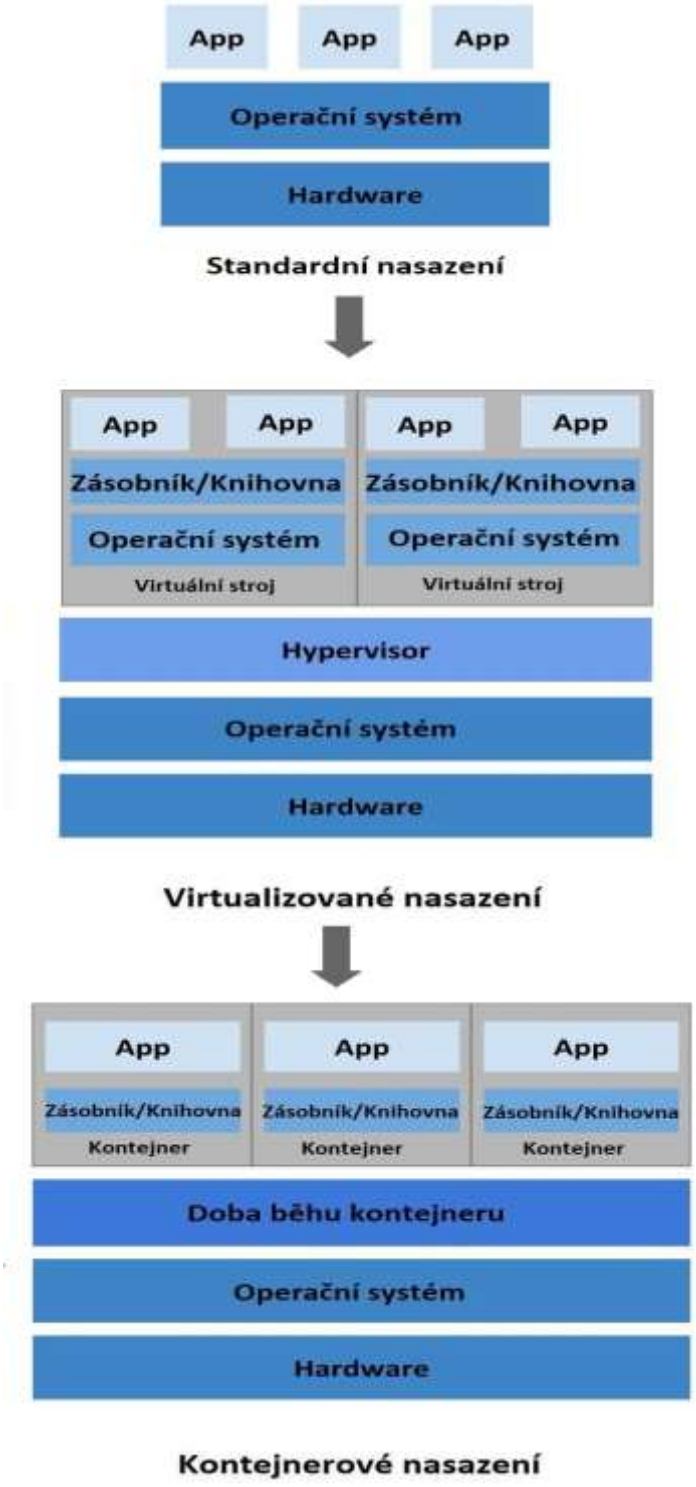
Ukázka kódu 1 Docker Compose YAML. Zdroj: [25]

Tento soubor YAML verze 2 vytvoří webovou aplikaci na portu 5000 s databází redis a se zdrojem `/var/log` a názvem oddílu `logvolume01`.

4.3 Kubernetes

Obrázek 7 Kubernetes logo. Zdroj: [26]

Projekt Kubernetes byl vytvořen a stále je z velké části vyvíjen a spravován společností Google. Jde o plnohodnotnou platformu pro správu Docker kontejnerů. Využívá se v mnoha produkčních prostředích a je vhodný i pro tvorbu hybridního klastru v rámci několika různých poskytovatelů. Jedná se o otevřený software, který podporuje automatické nasazení, škálování a správu kontejnerových aplikací. Je navržen podle ověřených principů, díky kterým Google provozuje více jak miliardu kontejnerů. Google tak navrhl otevřenou platformu, která je určena pro použití i u ostatních, konkurenčních poskytovatelů cloudu AWS a Microsoft Azure. Kubernetes dovoluje vývojářům nasadit aplikace do libovolné cloudové platformy, aniž by museli významně modifikovat aplikaci. Hlavní výhodou Kubernetes je samostatné nasazování kontejnerů, kdy se sleduje jejich dostupnost a efektivně se využívá veškerá výpočetní kapacita, která je k dispozici. To umožňuje udržet aplikaci co nejdostupnější. Dále vytváří abstrakci nad dostupným hardwarem a všechny připojené servery se tak chovají jako jeden stroj. Podobně jako Apache Mesos si tak řadí všechny fyzické zdroje do jednoho celku, ze kterého si podle potřeby postupně rozděluje zdroje. Avšak na rozdíl od ostatních orchestrátorů nabízí mnoho funkcí, je snadno rozšiřitelný a podporuje více kontejnerových řešení než Docker, Rocket, Windows kontejnery a další. Kubernetes seskupuje kontejnery do logických celků takzvaných podů. Jedná se o seskupení kontejnerů, které sdílí společné zdroje, nejčastěji je to síť a datové úložiště. Kubernetes poskytuje velké množství nastavení a je vhodný i pro velké projekty a datová centra [27].



Obrázek 8 Vývoj nasazení aplikací. Zdroj: [19]

Standardní doba nasazení

Na začátku organizace provozovaly aplikace na fyzických serverech. Neexistoval způsob, jak definovat hranice prostředků pro aplikace na fyzickém server, a to způsobilo problémy s přidělováním prostředků. Například pokud je na fyzickém serveru spuštěno více aplikací, mohou nastat případy, kdy by jedna aplikace zabírala většinu prostředků, a v důsledku toho by ostatní aplikace nedosahovaly dostatečné výkonosti. Řešením by bylo spuštění každé aplikace na jiném fyzickém serveru. To se však nezměnilo, protože zdroje byly nedostatečně využívány a pro organizace bylo drahé udržovat mnoho fyzických serverů [19].

Virtualizovaná doba nasazení

Jako řešení byla zavedena virtualizace. Virtualizace umožňuje spouštět více VM na jednom CPU fyzického server. Virtualizace dále umožňuje izolovat aplikace mezi VM a poskytuje úroveň zabezpečení, protože k informacím jedné aplikace nelze volně přistupovat jinou aplikací. Každý VM je stroj, na kterém běží všechny komponenty, včetně vlastního OS, na vrcholu virtualizovaného hardwaru [19].

Kontejnerová doba nasazení

Kontejnery jsou podobné VM, ale mají uvolněné izolační vlastnosti, které sdílejí OS mezi aplikacemi. Proto jsou kontejnery považovány za lehké. Podobně jako u VM má kontejner svůj vlastní systém souborů, procesor, paměť, procesní prostor a další. Protože jsou odděleny od základní infrastruktury, jsou přenosné přes cloud [19].

Co umožňuje Kubernetes

Kontejnery jsou dobrým způsobem, jak zabalit a spustit aplikace. V produkčním prostředí je třeba spravovat kontejnery, které spouštějí aplikace a zajistit, aby nedocházelo k prostojům. Pokud například kontejner selže, musí být spuštěn další kontejner. Kubernetes poskytuje framework pro spuštění distribuovaných systémů, stará se o škálování a správu služeb při selhání [19].

Kubernetes umožňuje vystavit kontejner pomocí DNS názvu, nebo pomocí své vlastní IP adresy. Pokud je síťový přenos do kontejneru vysoký, je schopen načíst rovnováhu a distribuovat síťový provoz tak, aby nasazení bylo stabilní. Kubernetes automaticky připojí úložný systém podle výběru, jako jsou místní úložiště, poskytovatelé veřejných

cloudů a další. Může vytvořit nové kontejnery pro nasazení, odstranit existující kontejnery a přizpůsobit všechny jejich zdroje novému kontejneru. Umožňuje určit kolik CPU a RAM každý kontejner potřebuje, když mají kontejnery specifikovány požadavky na zdroje, Kubernetes se může lépe rozhodovat o řízení zdrojů pro kontejnery. Umí restartovat kontejnery, které selhávají, nahrazuje a maže kontejnery, které neodpovídají stavu, který uživatel definoval. Umožňuje ukládat a spravovat citlivé informace, jako jsou hesla, tokeny a ssh klíče. Umí nasadit, aktualizovat a konfigurovat tyto informace bez odhalení konfigurace, aniž by se muselo znova sestavovat obrazy těchto kontejnerů [19].

Co Kubernetes neumožňuje

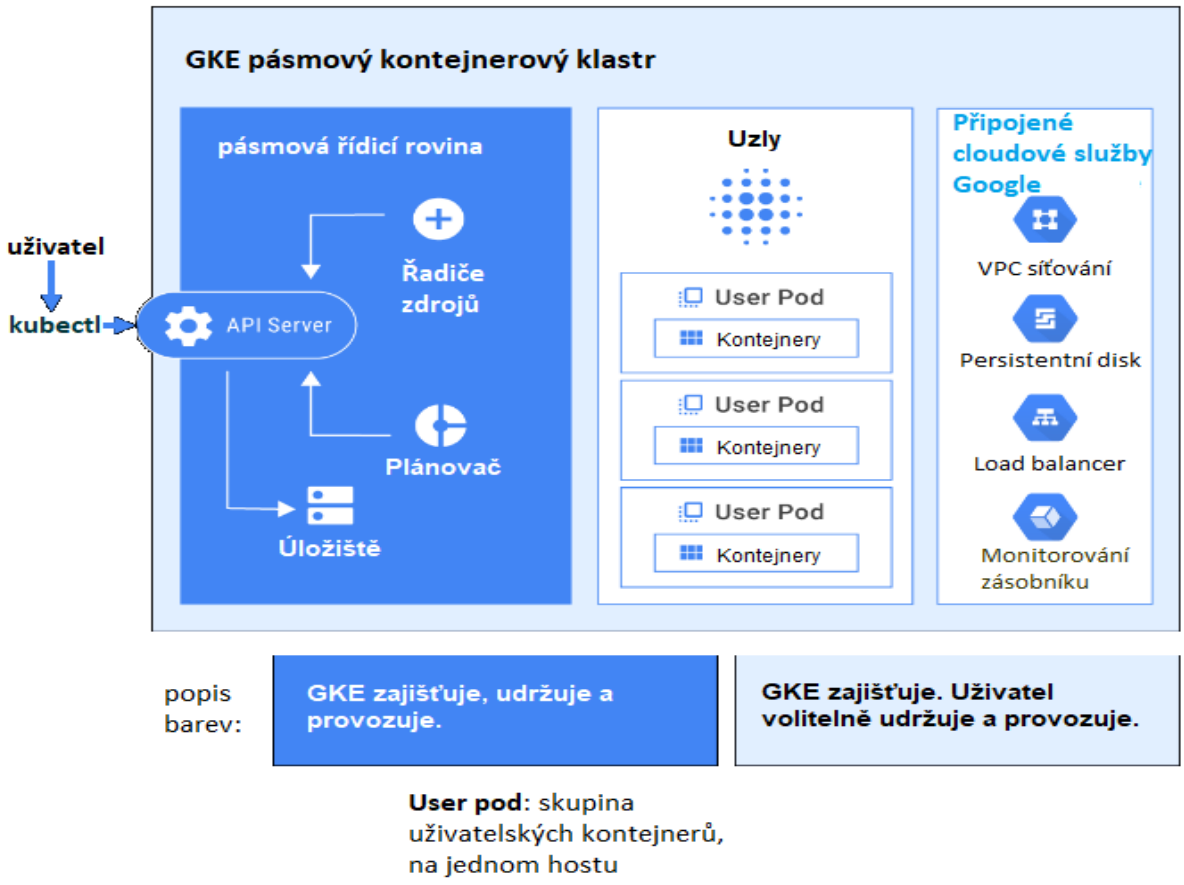
Kubernetes není tradiční komplexní systém PaaS (Platform as a Service = platforma jako služba), jelikož pracuje spíše na úrovni kontejneru než na úrovni hardwaru, poskytuje některé obecně použitelné funkce pro PaaS, jako je nasazení, škálování, vyrovnávání zatížení, protokolování a monitorování. Není monolitický a tato výchozí řešení jsou volitelná a připojitelná. Poskytuje základ vytváření platforem pro vývojáře, ale zachovává výběr uživatele a flexibilitu [19].

Neposkytuje služby na úrovni aplikací, rámců pro zpracování dat (Spark), databází (například MySQL), ani systémů úložiště klastrů (například Ceph). Takové komponenty mohou běžet na Kubernetes, nebo k nim může být přistupováno prostřednictvím přenosných mechanismů jako je Open Service Broker. Neumožňuje řešení protokolování, monitorování, nebo varování. Neposkytuje ani neřídí konfigurační jazyk nebo systém (například Jsonnet). Neposkytuje ani nepřijímá žádné komplexní konfigurace stroje, údržbu, správu, nebo samo-léčivé systémy. Kubernetes není pouhým orchestračním systémem, ve skutečnosti eliminuje potřebu orchestrace. Technická definice orchestrace je provedení definovaného pracovního postupu: nejprve proveďte A, poté B, potom C. Naproti tomu Kubernetes je tvořen sadou nezávislých, složitelných řídících procesů, které nepřetržitě posouvají aktuální stav k požadovanému stavu. Nezáleží na tom, jak se dostat z bodu A do bodu C. Centralizovaná kontrola není nutná, výsledkem je systém, který se snadněji používá, je výkonnější, robustnější, odolnější a rozšiřitelný [19].

Klastrová architektura

V Google Kubernetes Engine (GKE) se klastr skládá alespoň z jednoho hlavního stroje, který klastr řídí (cluster master) a více pracovních strojů zvaných uzly. Tyto hlavní a uzlové stroje používají systém orchestrace klastrů Kubernetes. Klastr je základem GKE: objekty Kubernetes, které přestavují kontejnerizované aplikace jsou spuštěny na vrcholu klastru [28].

Následující diagram poskytuje přehled architektury pro pásmový klastr v GKE [28].



Obrázek 9 **Pásmový klastr v GKE. Zdroj: [28]**

Cluster master a Kubernetes API

Cluster master spouští procesy řídicí roviny Kubernetes, včetně API serveru, plánovače a základních zdrojových prostředků API serveru. Životní cyklus mastera je spravován GKE, když se vytváří nebo odstraňuje klastr. To zahrnuje aktualizace Kubernetes spuštěného

na cluster masteru, který GKE provádí automaticky. Cluster master je sjednocený koncový bod pro klastr. Veškeré interakce s klastrem se provádějí prostřednictvím volání rozhraní Kubernetes API. Hlavní server spustí proces serveru Kubernetes API, aby tyto požadavky zpracoval. Volání rozhraní serveru lze provádět přímo přes http/gRPC, nebo nepřímo spuštěním příkazů od klienta příkazového řádku (kubectl) nebo interakcí s uživatelským rozhraním v cloudové konzoli. API server je rozbočovač pro veškerou komunikaci v klastru. Všechny interní klastrové procesy (jako jsou klastrové uzly, systém a součásti, aplikační řadiče) fungují jako klienti serveru API[28].

Interakce masteru a uzlu

Cluster master je zodpovědný za rozhodování o tom, co běží na všech uzlech klastru. To může zahrnovat plánování pracovních zátěží, jako jsou aplikace na kontejnerech a správu životního cyklu, škálování a vylepšování. Master také spravuje síťové a úložné prostředky pro tato pracovní zatížení. Master a uzly také komunikují pomocí rozhraní API Kubernetes[28].

5 Využití kontejnerové virtualizace pro analýzu velkých dat

Docker kontejnery poskytují způsob, jak zabalit aplikace se vším potřebným pro jejich spuštění, včetně základních obrazů OS, databází, knihoven a binárních kódů. Spuštěním Docker Enginu na hostitelském počítači, kontejnery Dockeru pracují výhradně s jádrem hostitelského OS, což znamená, že všechny kontejnerové aplikace fungují stejně, bez ohledu na základní infrastrukturu. Kromě toho může na jednom hostitelském počítači spouštět více aplikací, což vede k úsporám nákladů [29].

Izolace nástrojů pro velká data

Spolu s hardwarem používaným k nastavení a správě klastrů VD je sada nástrojů, které vývojáři a datoví vědci v oblasti dat používají k dokončení úloh zpracování nebo jiných úkolů pro VD. Problém, který často vzniká, je, že každý vývojář chce používat své vlastní specifické nástroje k tomu, aby udělal to, co potřebuje s daty, což vyžaduje distribuci celé škály nástrojů a jejich závislostí na každém stroji v klastru VD. U velkého počtu vývojářů rychle vyvstanou problémy se závislostí nástrojů a specifické požadavky jednoho nástroje mohou způsobit selhání jiného nástroje. Docker nabízí způsob, jak překonat tyto problémy se závislostí tím, že umožní vybudovat ekosystém VD, ve kterém je každý nástroj samostatný, spolu se všemi jeho náležitostmi. Vývojáři mohou používat své vlastní nástroje pro různé úlohy bez obav z konfliktu s jinými nástroji, protože každý nástroj je izolován v kontejneru [29].

Spuštění naplánovaných analytických úloh

Naplánovaná analytická úloha je typem úlohy automatizované manipulace s daty, která lze spustit buď v opakovaném plánu, nebo v určitém čase. Tyto typy úloh jsou velmi užitečné pro VD, která zaplavují organizace vysokou rychlostí, což vyžaduje určitou formu automatizace, aby udržovala krok s úkoly. Docker kontejnery mohou zvýšit efektivitu naplánovaných úloh tím, že umožní spouštět naplánované úlohy, aniž by je ručně nastavoval v každém uzlu v klastru VD [29].

Například Chronos je plánovač úloh odolný vůči chybám spuštěný na Apache Mesosu, který umožňuje spuštění instancí Dockeru do Mesos klastru a vytváří pro tyto instance naplánované analytické úlohy. V rámci Mesosu lze spouštět distribuované aplikace VD, jako je Hadoop nebo Spark [29].

S Chronos a Mesos mohou vývojáři nebo administrátoři systému naplánovat Docker kontejnery tak, aby spouštěly ETL (extract, transform, load = extrakce, transformace a nahrání dat), dávkové a analytické aplikace na opakujícím se, nebo časově specifickém základě, a to bez nutnosti ručního nastavení v uzlech klastru. Kromě výhody používání Dockeru pro plánovanou analýzu plánovač úloh Chronos také ukazuje graf závislosti úlohy, která pomůže sledovat závislosti různých úloh [29].

Poskytování prostředí pro vývoj velkých dat

Schopnost poskytovat prostředí VD na lokálním počítači je užitečné pro vývojáře, kteří se chtějí dozvědět více o různých technologiích a nástrojích potřebných k získání znalosti ekosystému VD. V kontextu vývoje je učení způsobem nejlepší způsob, jak získat znalosti. Docker může pomoci tím, že umožní vytváření klastru s více uzly na jednom hostitelském počítači a replikuje typické nastavení VD [29].

Například Ferry je nástroj, který umožňuje spouštět více uzlů kontejnerů na jednom hostitelském počítači pomocí Dockeru. To znamená, že vývojáři mohou definovat, provozovat a nasazovat balíčky VD pomocí standardu dat YAML čitelného pro člověka, nebo JSON. Například následující kód vytvoří zásobník velkých dat obsahující klastr s 5 uzly a jednoho klienta Linux, který bude komunikovat s Hadoopem [29]:

```
Backend:
-  storage:
personality: “hadoop“
instances“ 5
layers:
- “hive“
connectors:
- personality: “hadoop-client“Backend:
-  storage:
personality: “hadoop“
instances“ 5
```

Ukázka kódu 2 Zásobník Hadoopu. Zdroj: [29]

Po definování tohoto zásobníku VD jej lze spustit v Dockeru. Spuštěním příkazu `sudo ferry server` v terminálu Dockeru a následným startem Hadoopu [29].

Schopnost poskytovat lokální zásobník VD jako je tento, je užitečný pro vývojáře, kteří potřebují lokální prostředí pro vývojové účely, ale je také dobré pro datové vědce na experimentování s technologiemi VD a na rozvíjení svých znalostí [29].

Vybudování architektury velkých dat pro mikro služby

Docker usnadňuje přechod k budování architektury mikroslužeb pro aplikace VD. Mikroslužby jsou nezávislé, modulární služby a kontejnery Dockeru poskytují přirozenou platformu pro implementaci takového nastavení pro aplikace VD [29].

Mezi hlavní výhody mikroslužeb pro VD patří snadnější škálovatelnost aplikací a lepší kvalita dat. Použití VD vede k mnoha možným bodům selhání, které mohou vést ke snížení kvality dat, díky mikroslužbám mají vývojové týmy snazší práci při testování a údržbě služeb, což snižuje šance na nízkou kvalitu dat [29].

6 Návrh a realizace ukázkového řešení

6.1 Funkce skriptu

Pro praktickou část bakalářské práce je použitý bash skript, napsaný v Microsoft Visual Studio Code a testovaný na Ubuntu 20.04 LTS. Skript je vytvořený jako několik menu, které čekají na výběr uživatele, menu je udělané pomocí `while` cyklu, díky tomu je zamezeno vypnutí skriptu, při zadání nesprávné klávesy. Z hlavního menu je možné dostat se do nabídek Docker a Kubernetes, které mají vlastní menu s konkrétními možnostmi pro obě platformy, především instalace (ta je přístupná pouze, pokud se v systému Docker a Kubernetes nenachází) a odinstalování. Možnosti Docker jsou zvlášť rozděleny na správu kontejnerů, obrazů a Swarmu, z důvodu přehlednosti pro uživatele. Ve Swarmu má uživatel možnost inicializovat uzel správce a přes skript Swarm spravovat. Kubernetes menu má na výběr vytvoření cluster master, správu klastru a vytváření tokenů pro připojení se do klastru.

6.2 Realizace ukázkového řešení

Princip skriptu

Skript funguje pomocí `while` cyklu jako menu, které čeká na výběr od uživatele, z hlavního menu se lze dostat do nabídky Dockeru nebo Kubernetes. `While` cyklus se opakuje, dokud není zvolena správná možnost. Uživatel má možnost se do hlavního menu vrátit jak z nabídky Dockeru, tak z nabídky Kubernetes, a zároveň má volbu skript vypnout.

```
while [[ $usrin != "q" ]]
do
      #zavolání funkce vítejte
      Welcome
      if [[ $usrin == "d" ]]
            then
                  #vyprázdnění proměnné
                  docin=""
                  Docker
      elif [[ $usrin == "k" ]]
            then
                  #vyprázdnění proměnné
                  kubin=""
                  Kubernetes
    elif [[ $usrin != "q" ]]
            then
                  echo "Nesprávná klávesa."
    fi
done
```

Ukázka kódu 3 Princip skriptu. Zdroj: [autor]

Nabídka Dockeru

Nabídka Dockeru umožňuje nainstalovat všechny potřebné nástroje pro práci s Dockerem a zároveň je smazat. Po instalaci lze pomocí skriptu spouštět, zastavovat a mazat kontejnery, a to samé platí pro obrazy. Kontejnery a obrazy mají vlastní menu pro jejich správu, do kterého je možné se z menu Dockeru dostat.

Nabídka Docker Swarmu

Do Swarmu se lze dostat pouze, pokud je nainstalovaný Docker.

```
if ! [ -x "$(command -v docker)" ]
      then
            echo "Docker není nainstalován."
fi
      read -p "Pokud si přejete vytvořit Docker Machine a tam inicializovat Swarm, tak stiskněte 'dm', pokud přímo v systému, tak stiskněte 's'. :" docs
      if [[ $docs == "s" ]]
            then
            #práce se Swarmem v systému
            Dockerswarm
      elif [[ $docs == "dm" ]]
            then
            #práce se Swarmem pomocí Docker Machine
            Dockermachineswarm
      else
            echo "Nesprávná klávesa."
      fi
```

Ukázka kódu 4 Kontrola instalace Dockeru a výběr menu. Zdroj: [autor]

Pro funkci Docker Swarmu je ve skriptu využitý Docker Machine. Uživatel má volbu pro vytvoření vlastních Docker Machine, nebo předvoleného jednoho správcovského a dvou pracovních uzlů.

```
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER      ERRORS
manager1   -        virtualbox   Running   tcp://192.168.99.102:2376           v19.03.12
worker1    -        virtualbox   Running   tcp://192.168.99.103:2376           v19.03.12
worker2    -        virtualbox   Running   tcp://192.168.99.105:2376           v19.03.12
```

Obrázek 10 Docker Machine – jeden správce a dva pracovní uzly. Zdroj: [autor]

Po vytvoření těchto tří uzlů lze inicializovat Swarm na správcovském uzlu.

```
docker@manager1:~$ docker swarm init --advertise-addr 192.168.99.102
Swarm initialized: current node (o0ky42ey44cb48kghm7cmytvu) is now a manager.
```

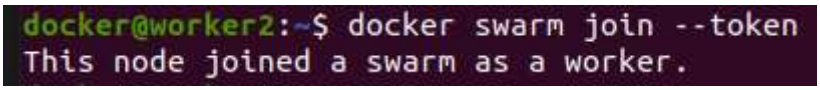
Obrázek 11 Inicializace Swarmu. Zdroj: [autor]

Do Swarmu se připojuje pomocí tokenu, který je třeba zadat do uzlu správce.

```
docker swarm join-token worker
```

Ukázka kódu 5 Připojení pracovního uzlu do Swarmu. Zdroj: [autor]

Tento příkaz vygeneruje token, který je třeba zadat do pracovního uzlu a pomocí toho se připojí do Swarmu. Stejným způsobem lze připojit i správcovský uzel, jen stačí v příkazu zaměnit `worker` za `manager`.



Obrázek 12 Přidání pracovního uzlu do Swarmu. Zdroj: [autor]

Po předchozích krocích je ve správci Swarmu možné spustit kontejnery. V následující ukázce je znázorněno nasazení kontejnerů do 3 uzlů ve Swarmu s pomocí webové služby nginx.

```
docker service create –replicas 3 -p 90:90 –name test1 nginx
```

Ukázka kódu 6 Spuštění kontejnerů ve Swarmu. Zdroj: [autor]

Obrázek 13 Spuštěné kontejnery ve Swarmu. Zdroj: [autor]

Pokud si uživatel nepřeje používat Docker Machine ke spuštění Swarmu, má možnost Swarm inicializovat přímo v OS. Inicializace Swarmu probíhá podobně jako v předchozím případě, nyní však stačí zadat pouze IP adresu správce.

```
if [[ $docs == "i" ]]
      then
      echo "Vaše IP adresa je. :"
     hostname -I
      read -p "Zadejte vaší IP adresu. :" docsin
     docker swarm init --advertise-addr $docsin
      echo "Po inicializaci Swarmu se zobrazí token, pomocí kterého
se lze přidat ostatní uzly do klastru."
```

Ukázka kódu 7 Inicializace Swarmu. Zdroj: [autor]

Obrázek 14 Servery ve Swarmu. Zdroj: [autor]

Ve Swarmu lze spustit kontejnery s výběrem počtu instancí, portu názvu a požadovaného obrazu.

```
elif [[ $docs == "s" ]]
      then
      read -p "Zadejte počet replik. :" docsr
      read -p "Zadejte port (80:80). :" docsp
      read -p "Zadejte váš název. :" docsn
      read -p "Zadejte název obrazu. :" docsi
      docker service create --replicas $docsr -p $docsp --name
$docsn $docsi
```

Ukázka kódu 8 Spuštění kontejneru ve Swarmu s výběrem. Zdroj: [autor]

Skript umožňuje škálování instancí, aby nedošlo k přetížení serverů.

```
elif [[ $docs == "sc" ]]
      then
            read -p "Zadejte název vaší Docker service. :" docsn
            read -p "Zadejte požadované škálování. :" docssc
            docker service scale $docsn=$docssc
```

Ukázka kódu 9 Škálování instancí ve Swarmu. Zdroj: [autor]

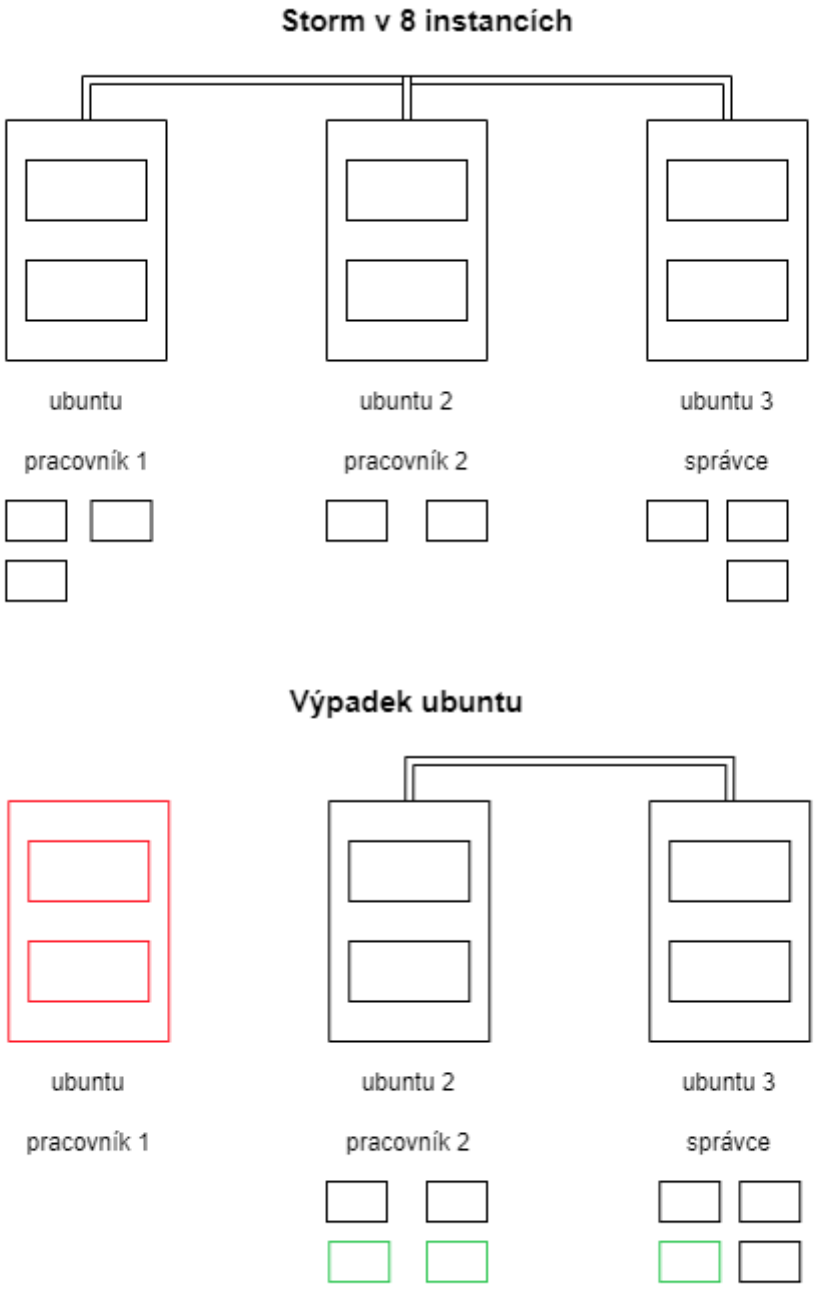
```
ID             NAME      IMAGE          NODE      DESIRED STATE  CURRENT STATE
5r07s1862cij   Storm.1   storm:latest   ubuntu2   Running        Running 8 seconds ago
q7f3gaspdwbs   Storm.2   storm:latest   ubuntu    Running        Running 22 seconds ago
moxfeo6ajx4v   Storm.3   storm:latest   ubuntu2   Running        Running 22 seconds ago
moxjjnbcivxn   Storm.4   storm:latest   ubuntu3   Running        Running 21 seconds ago
klyt0hncpki    Storm.5   storm:latest   ubuntu3   Running        Running 21 seconds ago
```

Obrázek 15 Škálování Stormu na 5 instancí, běžících na 3 serverech. Zdroj: [autor]

Při výpadku serveru se kontejnery přesunou automaticky na jiný server, v tom spočívá výhoda klastrování.

```
ID             NAME          IMAGE          NODE      DESIRED STATE  CURRENT STATE
22uuyt7hru0c   Storm.1       storm:latest   ubuntu3   Running        Running 11 seconds ago
o3sv4bysk6pd   Storm.2       storm:latest   ubuntu2   Running        Running 2 seconds ago
4a5kaoyhjhn0   Storm.3       storm:latest   ubuntu2   Running        Ready 4 seconds ago
8a5g2jb8heqw    \_ Storm.3   storm:latest   ubuntu    Shutdown       Running 36 seconds ago
rnptaa0b9fx1   Storm.4       storm:latest   ubuntu3   Running        Ready 4 seconds ago
oqbg8fr2tn7j    \_ Storm.4   storm:latest   ubuntu    Shutdown       Running 36 seconds ago
9pgfurk0g79c   Storm.5       storm:latest   ubuntu3   Running        Running 2 seconds ago
shak02zesnzx   Storm.6       storm:latest   ubuntu2   Running        Running 3 seconds ago
lkbyqyv5h5nw   Storm.7       storm:latest   ubuntu2   Running        Ready 4 seconds ago
jrv1vca7tohv    \_ Storm.7   storm:latest   ubuntu    Shutdown       Running 36 seconds ago
canz3xbkgx5j   Storm.8       storm:latest   ubuntu3   Running        Running 1 second ago
```

Obrázek 16 Výpadek serveru. Zdroj: [autor]



Obrázek 17 Graf výpadku serveru. Zdroj: [autor]

Nabídka Hadoopu

Skript umožňuje nasazení nástroje Docker Hadoop volbou při spouštění kontejnerů. Pokud již má uživatel Hadoop, tak skript nabídne jeho zastavení, nebo spuštění pomocí Docker Compose. V opačném případě proběhne následující část skriptu.

```
git clone https://github.com/big-data-europe/docker-hadoop.git
cd docker-hadoop
docker-compose up -d
docker exec -it namenode /bin/bash
hdfs dfs -mkdir -p /user/root
```

Ukázka kódu 10 Nasazení Docker Hadoopu. Zdroj: [autor]

Tímto se stáhne adresář s Docker Hadoopem, spustí se kontejnery. Běžící kontejnery má uživatel možnost zobrazit v menu s kontejnery.

Nabídka Kubernetes

V nabídce Kubernetes má uživatel možnost Kubernetes nainstalovat, nebo pokud se již v počítači nachází, tak odinstalovat. Kubernetes nemůže být nainstalovaný bez Dockeru a to z důvodu, že Docker je platforma pro kontejnerizaci a Kubernetes je orchestrátorem kontejnerových platforem, jako je Docker, tudíž bez něj nemůže fungovat. Ve skriptu je tento problém řešen následovně.

```
#kontrola zda je Kubernetes nainstalován
if [ -x "$(command -v kubeadm)" ];
     then
          echo "Kubernetes je již nainstalován."
     elif ! [ -x "$(command -v docker)" ]
          then
               echo "Pro instalaci Kubernetes musíte nejdříve
nainstalovat Docker."
#pokud není Kubernetes nainstalován a Docker je, tak instalace
Kubernetes
     else
```

Ukázka kódu 11 Podmínka pro instalaci Kubernetes. Zdroj: [autor]

Tato podmínka skriptu nastane, pokud si uživatel přeje nainstalovat Kubernetes. Nejdříve zkontroluje, zda již není nainstalován, pokud ano, tak podmínka končí a vrátí uživatele zpět do nabídky. Jestliže není nainstalován, tak zkontroluje, jestli je v systému přítomný Docker. Pokud jsou obě tyto podmínky splněny, tak skript nainstaluje Kubernetes.

Spuštění klastru v Kubernetes

Jestliže si uživatel přeje spustit klastr v Kubernetes, tak je nejdříve potřeba inicializovat Cluster master.

```
if [[ $kubm == "i" ]]
      then
            read -p "Zadejte vaší IP adresu. :" kubmin
            swapoff -a
            systemctl daemon-reload
            systemctl restart kubelet
            kubeadm init --pod-network-cidr=192.168.0.0/16 -
      apiserver advertise-address=$kubmin --ignore-preflight-
      errors=
            echo "Vygenerovaný token použijte pro připojení
      pracovníka do klastru."
```

Ukázka kódu 12 Inicializace klastru v Kubernetes. Zdroj: [autor]

Při volbě spouštění klastru v Kubernetes je třeba zadat lokální IP adresu, která lze zjistit z hlavního menu. Dále skript vypne swapoff, s kterým není možné klastr spustit, protože při spouštění uzlů se swapoff ztrácí mnoho izolačních vlastností, díky nimž je sdílení počítačů možné. Restartují se potřebné systémy pro klastr a inicializuje se master a po jeho inicializaci se objeví vygenerovaný token.

```
kubeadm join 192.168.0.123:6443 --token nyjrly.s13nsfczyp8vem14 \
    --discovery-token-ca-cert-hash sha256:402cad7177f820b88243ad2075792129101b9553a1df0d5c31c4550e06d5e252
```

Obrázek 18 Ukázkový token pro připojení do klastru. Zdroj: [autor]

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

Obrázek 19 Připojení uzlu do klastru. Zdroj: [autor]

Pomocí tokenu lze připojit pracovníka do klastru. Token expiruje po 24 hodinách, poté je potřeba vytvořit nový token. Při ztrátě tokenu skript umožňuje zobrazit list tokenů, nebo vygenerovat nový token.

```
read -p "Pokud si přejete vytvořit token, stiskněte 'v', pro token
list 'l'. :" kubmin
if [[ $kubmin == "v" ]]
     then
          kubeadm token create
 elif [[ $kubmin == "l" ]]
     then
          kubeadm token list
else
     echo "Nesprávná klávesa."
fi
```

Ukázka kódu 13 Vytvoření a zobrazení tokenu. Zdroj: [autor]

7 Závěr

Cílem této bakalářské práce bylo popsat problematiku kontejnerové virtualizace, její možnosti využití v problematice analýzy velkých dat a následně implementovat skript pro UNIX.

Součástí teoretické části bylo seznámení s velkými daty, metody analýzy velkých dat a také kontejnerizačními a virtualizačními technologiemi. Popsané metody kontejnerizace a virtualizace byly poté aplikovány v praktické části.

V rámci praktické části byl vytvořen skript pro UNIX. Při implementaci skriptu byly využity kontejnerizační technologie Docker, Kubernetes a nástroj Hadoop. Byla použitá klastrová technologie Docker Swarmu a klastry Kubernetes.

Pomocí skriptu je možné spravovat kontejnery a obrazy a také sledovat jejich stav. Skript je rozdělený na dvě části, v první části je možné se dostat do nabídky s Dockerem, Docker Swarmem a jejich nasazení do systému. Druhá část se věnuje Kubernetes, s kterou lze pracovat pouze po instalaci Dockeru.

Skript by mohl sloužit pro uživatele, kteří chtějí využívat kontejnerové platformy Docker a Kubernetes, umožňuje jejich jednoduchou instalaci a nasazení klastrů do obou virtualizačních prostředků. Klastry je možné pomocí skriptu sledovat, upravovat, škálovat a mazat jednotlivé uzly z uzlu správce. Pomocí skriptu je možné nasazovat, sledovat a odstraňovat kontejnery a obrazy. Hlavním významem je snazší používání kontejnerizačních technologií, místo zadávání několika příkazů do příkazového řádku, stačí pouze zvolit v menu, co si uživatel přeje, a to skript provede.

Mezi plánované budoucí rozšíření je upravení skriptu, aby bylo možné skript spouštět také ve Windows a vytvoření GUI pro lepší uživatelskou přehlednost a větší komfort.

8 Seznam použité literatury

[1] CHEN, Min, Shiwen MAO a Yunhao LIU. Big Data: A Survey. *Mobile Networks and Applications* [online]. 2014, **19**(2), 171–209. ISSN 1383-469X, 1572-8153. Dostupné z: doi:10.1007/s11036-013-0489-0

[2] RUSSOM, Philip. Big Data Analytics. *BIG DATA ANALYTICS*. nedatováno, 38.

[3] *What Is Big Data? | Oracle* [online]. [vid. 2020-03-22]. Dostupné z: https://www.oracle.com/big-data/guide/what-is-big-data.html#link1

[4] YEOMAN, Ian. Big Data. *Journal of Revenue and Pricing Management* [online]. 2019, **18**(1), 1–1. ISSN 1477-657X. Dostupné z: doi:10.1057/s41272-019-00191-9

[5] LI, Gang. Big data related technologies, challenges and future prospects: Min Chen, Shiwen Mao, Yin Zhang and Victor C. M. Leung New York: Springer, 2014 ISBN 978-3-319-06244-0 (printed book), 978-3-319-06245-7 (e-book), 89 pp, USD54.99 (printed book), USD39.99 (e-book). *Information Technology & Tourism* [online]. 2015, **15**(3), 283–285. ISSN 1098-3058, 1943-4294. Dostupné z: doi:10.1007/s40558-015-0027-y

[6] MAYER-SCHÖNBERGER, Viktor a Kenneth CUKIER. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. B.m.: Houghton Mifflin Harcourt, 2013. ISBN 978-0-544-00293-7.

[7] CAVANILLAS, Jose. *New horizons for a data-driven economy: a roadmap for usage and exploitation of big data in Europe*. New York, NY: Springer Berlin Heidelberg, 2015. ISBN 978-3-319-21568-6.

[8] *Soubor:Hadoop logo.svg – Wikipedie* [online]. [vid. 2020-03-18]. Dostupné z: https://commons.wikimedia.org/wiki/File:Hadoop_logo.svg

[9] *Apache Hadoop* [online]. [vid. 2019-11-01]. Dostupné z: https://hadoop.apache.org/

[10] FOUNDATION, Apache software. *Français : Logo de la plateforme de traitement de données Apache Spark* [online]. 2013 [vid. 2020-03-18]. Dostupné z: https://commons.wikimedia.org/wiki/File:Apache_Spark_logo.svg

[11] VERMA, Amit. Top 10 Open Source Big Data Tools in 2019 [Updated]. *Whizlabs Blog* [online]. 12. březen 2018 [vid. 2019-12-02]. Dostupné z: https://www.whizlabs.com/blog/big-data-tools/

[12] *Apache Storm - Bauman National Library* [online]. [vid. 2020-03-18]. Dostupné z: https://en.bmstu.wiki/Apache_Storm

[13] *Apache Storm* [online]. [vid. 2019-12-03]. Dostupné z: https://storm.apache.org/

[14] *Apache Storm - Introduction - Tutorialspoint* [online]. [vid. 2020-07-17]. Dostupné z: https://www.tutorialspoint.com/apache_storm/apache_storm_introduction.htm

[15] What Are Containers (Container-based Virtualization or Containerization)? *SearchITOperations* [online]. [vid. 2020-03-26]. Dostupné z: https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization

[16] Container-Based Visualization. *SolutionDots* [online]. 30. duben 2018 [vid. 2019-07-02]. Dostupné z: https://solutiondots.com/blog/technology-blog/container-based-virtualization-depth/

[17] What is a Container? *Docker* [online]. [vid. 2019-06-14]. Dostupné z: https://www.docker.com/resources/what-container

[18] *Embedded operating systems*. New York, NY: Springer Berlin Heidelberg, 2018. ISBN 978-3-319-72976-3.

[19] *What is Kubernetes* [online]. [vid. 2019-08-01]. Dostupné z: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[20] *What is a Virtual Machine and How Does it Work | Microsoft Azure* [online]. [vid. 2020-03-23]. Dostupné z: https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/

[21] *Virtualization 101: What is a Hypervisor?* [online]. [vid. 2019-07-31]. Dostupné z: https://www.pluralsight.com/blog/it-ops/what-is-hypervisor

[22] NIEDRINGHAUS, Paige. Docker 101: Fundamentals & The Dockerfile. *Medium* [online]. 30. duben 2019 [vid. 2019-11-18]. Dostupné z: https://itnext.io/docker-101-fundamentals-the-dockerfile-b33b59d0f14b

[23] What is Docker? *Opensource.com* [online]. [vid. 2019-08-01]. Dostupné z: https://opensource.com/resources/what-docker

[24] What is Docker Swarm? *Sumo Logic* [online]. [vid. 2020-02-16]. Dostupné z: https://www.sumologic.com/glossary/docker-swarm/

[25] Overview of Docker Compose. *Docker Documentation* [online]. 20. únor 2020 [vid. 2020-02-22]. Dostupné z: https://docs.docker.com/compose/

[26] LUIGI, Author. An Introduction to Kubernetes for Data Scientists. *ML in Production* [online]. 29. duben 2019 [vid. 2019-11-18]. Dostupné z: https://mlinproduction.com/intro-to-kubernetes/

[27] 4 Reasons to use Apache Mesos Frameworks. *Container Solutions* [online]. 29. září 2015 [vid. 2019-06-14]. Dostupné z: https://container-solutions.com/reasons-use-apache-mesos-frameworks/

[28] Cluster architecture | Kubernetes Engine Documentation. *Google Cloud* [online]. [vid. 2020-02-22]. Dostupné z: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture?hl=cs

[29] Docker use cases - How to handle big data with Docker. *Big Data Made Simple* [online]. 11. květen 2018 [vid. 2020-02-13]. Dostupné z: https://bigdata-madesimple.com/docker-use-cases-how-to-handle-big-data-with-docker/

[30] *What is a Bash Script? - Bash Scripting Tutorial* [online]. [vid. 2020-07-17]. Dostupné z: https://ryanstutorials.net/bash-scripting-tutorial/bash-script.php

[31] *Bash Scripting Tutorial for Beginners - LinuxConfig.org* [online]. [vid. 2020-07-17]. Dostupné z: https://linuxconfig.org/bash-scripting-tutorial-for-beginners

[32] *Scripting language* [online]. 2020 [vid. 2020-07-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Scripting_language&oldid=967779150

9 Seznam obrázků

10 Seznam ukázek kódu

Zadání bakalářské práce

Autor:	**David Vondráček**
Studium:	I1600633
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Big data a kontejnerová virtualizace**
Název bakalářské práce AJ:	Big Data and Container Virtualization

Cíl, metody, literatura, předpoklady:

Cílem práce je popsat problematiku kontejnerové virtualizace a její možnosti využití v problematice analýzy velkých dat. Student v práci představí kontejnerovou virtualizaci a provede rešerši dostupných kontejnerizačních řešení, jak je možné využít kontejnerizaci v analýze velkých dat. Dále student navrhne a implementuje možné ukázkové řešení s využitím kontejnerizace pro analýzu velkých dat. Osnova: 1. Úvod 2. Big data a analýza velkých objemů dat 3. Nástroje pro zpracování velkých dat 4. Kontejnerová virtualizace 5. Využití kontejnerové virtualizace pro analýzu velkých dat 6. Návrh a realizace ukázkového řešení 7. Závěr

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Michal Macinka
Datum zadání závěrečné práce:	14.1.2018