Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Využití Dockeru pro správu aplikací v kontejnerech

Diplomová práce

Autor: Tomáš Burda
Studijní obor: AI2

Vedoucí práce: doc. RNDr. Petra Poulová, Ph.D.

Odborný konzultant: Ing. David Petrík
Systemart s. r. o.

Hradec Králové duben 2020

Prohlášení:

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

Burda

V Hradci Králové dne 26.4.2020 Tomáš Burda

Poděkování:

Děkuji vedoucí diplomové práce doc. RNDr. Petře Poulové, Ph.D. za metodické vedení práce a Ing. Davidu Petríkovi ze společnosti Systemart s. r. o. za odborný dohled a užitečné rady.

Anotace

Tato diplomová práce se zabývá kontejnerizací aplikací za využití moderních technologií. Kontejnerizace aplikací je v současnosti populárním trendem, který přináší firmám řadu výhod, zejména jednoduchý přesun aplikací do cloudového prostředí. Mezi cíle práce patří seznámení čtenářů s problematikou kontejnerizace, o které doposud v češtině nevyšlo příliš publikací. Dále vysvětluje využití moderních technologií, jako je Docker nebo Kubernetes, což jsou v oblasti kontejnerizace v současnosti hojně využívané technologie. Praktická část obsahuje konkrétní případ využití výše zmíněných technologií, a sice kontejnerizaci aplikace Logeto a veškerých jejích komponent, při současné automatizaci celého procesu. Praktické využití je také jedním z hlavních cílů této práce. Kontejnerizace aplikace a automatizace celého procesu totiž výrazně snižuje dobu nasazení nové verze, což minimalizuje i dopad na koncové uživatele. Přínos práce tedy spočívá nejen v detailním popisu procesu kontejnerizace, ale i v praktické ukázce využití technologie v plně funkční webové aplikaci.

Klíčová slova

Cloud computing, Kontejnery, Dockerfile, Docker, Kubernetes, Cluster

Annotation

Title: Using Docker to manage applications in containers

The thesis deals with containerization of applications by using modern technologies. Containerization of applications is currently a popular trend, which brings a number of benefits to companies, especially simple moving applications to the cloud environment. The aim of this work is to acquaint readers with the issue of containerization, which has not been published plentifully in Czech so far. Furthermore, the thesis explains the use of modern technologies, such as Docker or Kubernetes, which are currently widely used technologies in this field. The practical part contains a particular case of utilization of the above-mentioned technologies, namely the containerization of the Logeto application including all its components and simultaneous automation of the whole process. Practical use is also one of the main objectives of this thesis. Containerization of the application and automation of the whole process significantly reduces the time of implementation of the new version, which minimizes the impact on end users. The benefit of this thesis therefore lay not only in a detailed description of the containerization process, but also in a practical demonstration of the use of this technology in a fully functional web application.

Key words

Cloud computing, Containers, Dockerfile, Docker, Kubernetes, Cluster

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam grafů

Seznam tabulek

1 Úvod

Tato diplomová práce se zabývá kontejnerizací aplikací s využitím moderních technologií jako jsou Docker nebo Kubernetes. Docker je v poslední době velmi využívaná platforma pro vývoj, nasazení a spuštění aplikací v kontejnerech. Díky této platformě je možné aplikace izolovat od infrastruktury a spouštět je v různých prostředích. Zároveň Docker usnadňuje a urychluje i samotný vývoj aplikací.

V první části práce jsou definovány teoretické předpoklady pro praktické využití těchto moderních technologií, které umožnily společnostem v dnešní době jednoduchý přesun aplikací z vlastní fyzické infrastruktury do cloudu. Důvodem k přesunu je odstranění komplikací při správě vlastní infrastruktury a vysoké počáteční náklady za hardware a software. Proto je cloud computing popsán již v úvodní kapitole.

V minulosti byl přesun aplikací do cloudu složitý. Bylo třeba prostředí (vývojové, testovací a produkční) vždy správně nakonfigurovat se vším, co je potřeba ke správnému fungování aplikace. Vše se ale změnilo s rozvojem kontejnerů a výše zmíněných technologií. Kontejnery totiž poskytují prostředí izolované od okolí, které obsahuje všechny závislosti pro spuštění aplikace. Tím se usnadnil přesun aplikací mezi prostředími, která nejsou totožná. Kontejnery zároveň pomáhají snižovat konflikty mezi aplikacemi provozovanými na stejné infrastruktuře.

Kontejnery lze jednoduše spravovat na platformě Docker, v případě velkého množství kontejnerů je ale správa komplikovanější a vyžaduje nějakou formu automatizace. K tomuto účelu se používají nástroje pro orchestraci kontejnerů. Tyto nástroje jsou detailně popsány v závěru teoretické části.

Praktická část se zabývá procesem kontejnerizace již existující webové aplikace Logeto a veškerých jejích komponent, které jsou vyvíjeny společností Systemart s.r.o. Aplikace a její komponenty jsou založeny na technologiích .Net Frameworku, a proto byly použity Windows i Linux kontejnery. Kontejnerizace těchto aplikací je prvním krokem k modernizaci jejich architektury a migraci do cloudu. Za celý proces kontejnerizace aplikace Logeto zodpovídá autor diplomové práce pod dohledem vedoucího vývoje.

2 Cíl práce

Prvním cílem této práce je seznámení s problematikou kontejnerizačních technologií. Dalším z cílů je kontejnerizace aplikace Logeto a její nasazení do nástroje pro orchestraci kontejnerů Kubernetes, který je nakonfigurován na lokálních virtuálních strojích. Posledním cílem je automatizace jak tohoto procesu, tak i nasazení kontejnerizované aplikace do vývojového, testovacího a produkčního prostředí v Kubernetes.

I. Teoretická část

3 Cloud computing

Cloud computing (cloud) v současnosti představuje jeden z nejvýznamnějších trendů v informačních technologiích. Tento trend napomohl i k růstu využívání nástrojů pro správu kontejnerů, jako je například Docker nebo Kubernetes. Kontejnery změnily způsob, jakým jsou dnes aplikace vyvíjeny a provozovány a výrazně zjednodušily přesun aplikací právě do cloudu. [1]

Cloud computing umožňuje přístup k výpočetním prostředkům a službám na vyžádání prostřednictvím internetu. Uživatelé k nim mohou přistupovat vzdáleně a odkudkoliv, zároveň platí pouze za ty služby, které skutečně využívají. [2] Služby pokrývají celou řadu možností, od pronájmu úložiště nebo virtuálního stroje, přes zpracování přirozeného jazyka umělou inteligencí, až po standardní kancelářské aplikace. Téměř každá služba, která nevyžaduje fyzický přístup k hardwaru počítače, může být poskytována prostřednictvím cloudu. Mezi charakteristické vlastnosti cloud computingu patří [3]:

- **Samospráva a poskytnutí na vyžádání** – Výpočetní prostředky, které uživatelé využívají, jsou poskytnuty automaticky, bez potřeby lidské aktivity ze strany poskytovatele.
- **Škálovatelnost a elasticita** – Využití prostředků je možné rychle a snadno vytvářet, zvyšovat, přidávat nebo snižovat. Nepotřebné prostředky mohou být zase rychle uvolněny.
- **Měřená služba** – Umožnuje sledování spotřebovaných výpočetních prostředků a služeb.
- **Přístup přes internet** – Výpočetní prostředí je dostupné přes internet z jakékoliv části světa a z jakéhokoliv zařízení.
- **Multitenancy** – Výpočetní prostředí je sdíleno mezi více uživatelů, nicméně je zajištěno vzájemné oddělení a izolace, aby uživatelé neměli přístup k informacím a prostředkům ostatních uživatelů.

Výhodou využívání služeb cloud computingu je to, že společnosti se mohou vyhnout prvotním nákladům za pořízení a údržbu vlastní infrastruktury. Poskytovatelé služeb zase mohou těžit z významných úspor z rozsahu poskytování stejných služeb širokému okruhu zákazníků. Nevýhodou mohou být náklady spojené s přesunem stávajícího řešení do cloudu.

Všechny cloudové služby nejsou stejné a neexistuje žádný typ cloud computingu, který by byl vhodný pro všechny. Cloud computing dělíme podle toho, jak je poskytován, a podle služby, kterou poskytuje. [4]

3.1 Modely nasazení

Modely nasazení rozlišují cloud podle místa, ve kterém je umístěna infrastruktura pro nasazení služeb, a dle toho, kdo má nad touto infrastrukturou kontrolu. [2][5][6]

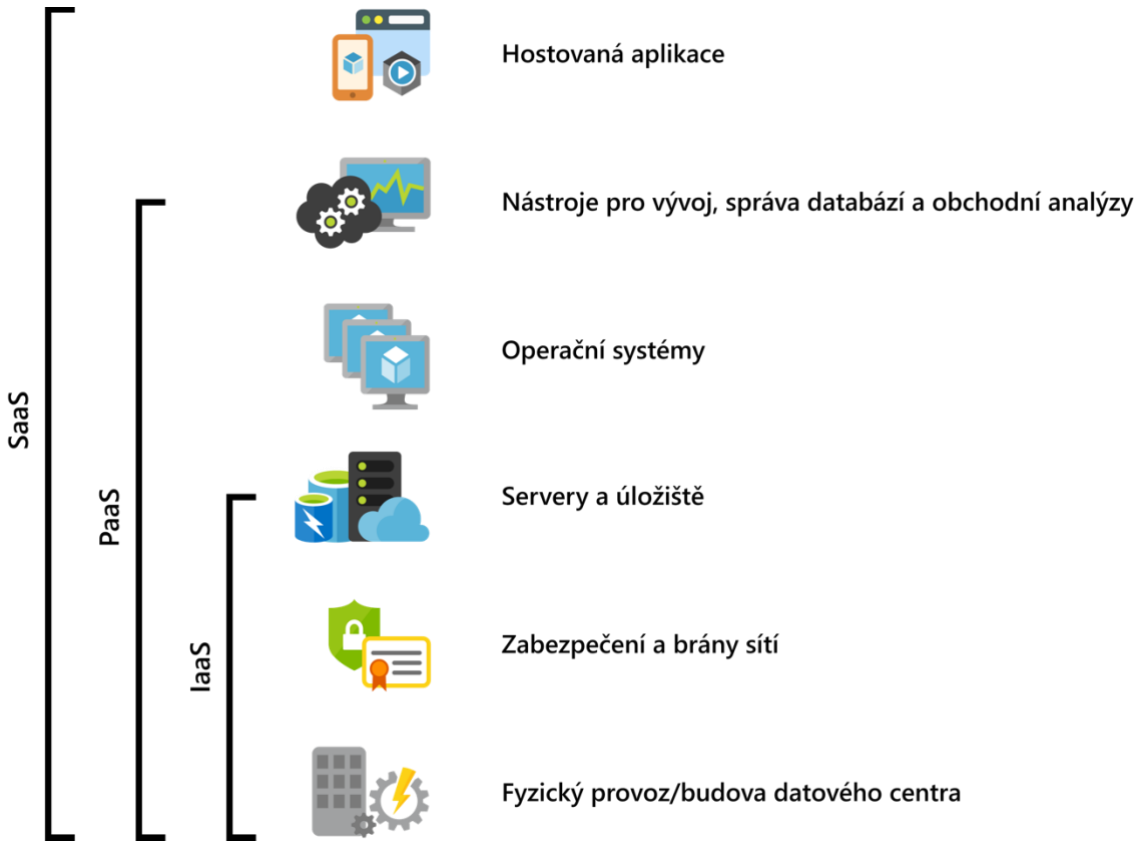
- **Veřejný cloud** – Cloudová infrastruktura a služby jsou vlastněny a provozovány v prostorách poskytovatele cloudu a jsou veřejně poskytovány jakýmkoliv uživatelům. Koncoví uživatelé mají přístup ke službám prostřednictvím veřejné sítě, jako je internet. Veřejný cloud nabízí vysokou škálovatelnost, automatickou údržbu, ale může být zranitelnější vůči útokům, kvůli své dostupnosti širokému okruhu uživatelů. Mezi běžné poskytovatele veřejného cloudu patří Microsoft Azure nebo Amazon Web Services.
- **Privátní cloud** – Cloudová infrastruktura je využívána a vlastněna výhradně jednou společností. Společnost může cloud spravovat a přizpůsobit podle svých potřeb. Může být hostován v lokalitě organizace, nebo v datovém centru u některého poskytovatele cloudu. Privátní cloud poskytuje nejvyšší úroveň zabezpečení a kontroly. Nevýhodou jsou vysoké vstupní náklady, z tohoto důvodu je vhodný pro větší společnosti. Poskytovatelé privátního cloudu jsou VMware a OpenStack.
- **Hybridní cloud** – Kombinuje veřejný a privátní cloud. Tito jsou propojeny standardizovanou nebo patentovanou technologií, která umožňuje mezi nimi sdílet data a aplikace. Společnosti hostují své důležité aplikace na svých

vlastních serverech, aby získaly vyšší úroveň zabezpečení, zatímco sekundární aplikace jsou umístěny na serverech u poskytovatele veřejného cloudu.

- **Komunitní cloud** – Cloudová infrastruktura je sdílena mezi více organizací, které sdílejí stejný obor zájmu, bezpečností požadavky nebo předpisy. Může jej vlastnit, spravovat a provozovat jedna nebo více organizací v komunitě.

3.2 Modely služeb

Většinu služeb cloud computingu lze rozdělit do třech hlavních typů, ke kterým mají uživatelé přístup prostřednictvím internetu. Někdy jsou označovány jako *stack* pro cloud computing, protože jsou postaveny jedna na druhé. [6][7]



Obr. 1 Modely služeb
Zdroj: [6]

- **Infrastructure as a Service (IaaS)** – Jedná se o nejzákladnější typ služby cloud computingu. V této službě poskytovatel cloudu poskytuje uživatelům přístup ke službám infrastruktury, jako jsou virtuální stroje, databázové systémy nebo uložiště, na kterých uživatelé provozují své aplikace. Uživatelům pomáhá vyhnout se složité správě vlastní infrastruktury. Mezi poskytovatele těchto služeb patří Microsoft Azure, Amazon Web Services nebo Google Cloud.
- **Platform as a Service (PaaS)** – Služba poskytuje přístup k vývojářským nástrojům, které vývojářům usnadňují vytvářet webové a mobilní aplikace. Stejně jako IaaS zahrnuje infrastrukturu (servery, uložiště a síťové prvky), ale navíc také middleware, systémy pro správu databází a další nástroje. Je navržena tak, aby podporovala všechny etapy životního cyklu aplikace (sestavení, testování, nasazení, správu a aktualizaci). Uživatelé se mohou vyhnout nákupu a správě softwarových licencí, podpůrné infrastruktury nebo nástrojů pro správu kontejnerů, jako je Docker nebo Kubernetes. Příkladem služby může být Azure Kubernetes Service (AKS).
- **Software as a Service (SaaS)** – Umožňuje uživatelům připojit se ke cloudovým aplikacím a používat je přes internet pomocí webového prohlížeče nebo mobilní aplikace. Obvyklými příklady jsou sdílené kalendáře, e-mail nebo Microsoft Office 365. Veškerá podpůrná infrastruktura, middleware, software a data aplikace jsou umístěna v datovém centru u poskytovatele služeb. Poskytovatel služeb spravuje software a hardware. Zajišťuje také dostupnost a zabezpečení aplikace a dat.

4 Kontejnery

Kontejnery jsou softwarové balíčky, které poskytují prostředí pro nasazení a spuštění aplikací. Obsahují tedy zdrojové kódy se všemi závislostmi, systémové knihovny, systémové nástroje a konfigurační soubory, které jsou potřebné ke spuštění aplikací uvnitř kontejnerů. Jedná se o tzv. kontejnerizované aplikace. Běžné aplikace jsou oproti kontejnerizovaným aplikacím nainstalovány a spuštěny přímo na hostitelském operačním systému. Problémem běžných aplikací je, že nelze spustit stejné aplikace v různých verzích souběžně z důvodu konfliktů. Využívají totiž stejný souborový systém, síťové rozhraní, porty a další.

Kontejnerizované aplikace mohou uvnitř kontejneru provádět jakoukoliv operaci, která nebude mít vliv na ostatní běžící kontejnery. Jejich další vlastností je izolace softwaru od okolí, čímž právě pomáhají snižovat konflikty mezi aplikacemi nebo vývojářskými týmy, které používají různé programy na stejné infrastruktuře. Kontejnery také řeší problém s přesunem softwaru z jednoho prostředí do druhého, během kterého může nastat několik problémů, pokud prostředí nejsou totožná. Může se jednat o přesun aplikace z lokálního do produkčního prostředí, z fyzického stroje na virtuální počítač, nebo z jedné platformy cloudu na druhou.

Kontejnery dávají vývojářům možnost rychleji vyvíjet a dodávat libovolné aplikace napsané v různých programovacích jazycích a umožňují vytvářet infrastrukturu, kterou je jednodušší aktualizovat a udržovat. Využití naleznou zejména pro vývoj mikroslužeb a pro zvýšení DevOps produktivity, jelikož snižují časovou náročnost na ladění a diagnostiku rozdílů mezi prostředími, a tím pádem zbývá více času na vývoj, testování a nasazení aplikací.

Nástroje pro správu kontejnerů, které budou detailně popsány v dalších kapitolách jsou Docker Enterprise Edition, Docker Compose, Docker Swarm nebo Kubernetes, které poskytují mechanismy pro nasazení, údržbu a škálování kontejnerizovaných aplikací na zařízeních s operačním systémem Linux nebo Windows a na různých cloudových platformách (Microsoft Azure, Google Cloud nebo Amazon Web Services). [1][9][10]

4.1 Historie kontejnerů

Spousta nových technologií vychází z nápadů, které již existovaly v minulém století. Stejně tak i kontejnery, které patří mezi populární technologie v dnešní době, vychází z práce vytvořené před 40-ti lety. Je možné tedy tvrdit, že historie kontejnerů začíná u jednoduchého systému, který byl součástí unixového jádra v sedmdesátých letech, a končí u moderních nástrojů pro správu kontejnerů, které využívají firmy jako Google, Microsoft nebo Facebook.

Vznik dnešních kontejnerů lze sledovat již od roku 1979, kdy byl představen chroot, který umožnil změnit kořenový adresář pro daný proces i jeho potomky. Tento postup byl počátkem izolace a běžně se používal k ochraně operačního systému. Izoloval aplikace a procesy jako jsou FTP, Apache, BIND nebo Sendmail, které jsou veřejně dostupné a mohou být potencionálně zneužity. V případě zneužití chyby v aplikaci, díky které se útočník dostane do počítače, pak neohrozí celý systém, protože uvidí pouze soubory, které využívá konkrétní aplikace v chroot prostředí. Chroot je ovšem velmi jednoduchý a má svá omezení. [11]

To se změnilo v roce 2000, kdy byl zveřejněn FreeBSD 4.0 s novým příkazem jail. Tento příkaz rozšiřuje schopnosti chrootu, a proto byl navržen tak, aby umožnil poskytovatelům hostingů snadno a bezpečně rozdělit počítačový systém do několika nezávislých menších systémů, tzv. jails, s možností přidělení IP adresy pro každý systém a konfiguraci.

V roce 2004 společnost Sun Microsystems (nyní Oracle) vydala první verzi systému Solaris 10, která zahrnovala Solaris Containers, ze kterých se později vyvinulo Solaris Zones. Jednalo se o první komerční implementaci kontejnerové technologie, která je stále využívána pro podporu kontejnerových implementací. Konečně v roce 2008 byly zveřejněny Linux Containers (LSX) ve verzi linuxového jádra 2.6.24. Největší popularity si Linux kontejnery ale získaly až po roce 2013, kdy byla vydána verze linuxového jádra 3.8 a o několik měsíců později vznikla technologie Docker. Tato technologie ukázala, jakým způsobem může být práce s aplikacemi v kontejnerech výrazně usnadněna.

Společnosti jako Google, které musí řešit škálování aplikací, začaly naléhat na vývoj kontejnerových technologií už na začátku 21. století, aby si usnadnily

distribuci svých aplikací napříč datovými centry po celém světě. Několik společností si vyvíjelo vlastní linuxové jádro s podporou kontejnerů pro vlastní užívání. S rostoucí potřebou kontejnerových technologií, začal i Google některé vlastní technologie přidávat do linuxového jádra. [10]

V roce 2016 ve spolupráci s Dockerem představila také společnost Microsoft, své kontejnerové řešení Windows Containers, které se dělí podle způsobu izolace na dva typy kontejnerů Windows Server Containers a Hyper-V Containers.

Windows Server Containers jsou obdobou Linux kontejnerů, které sdílejí jádro hostitelského operačního systému se všemi běžícími kontejnery. Tato izolace je dostupná pouze na Windows Serveru. Oproti tomu Hyper-V Containers vytvoří pro každý kontejner optimalizovaný virtuální počítač. Tento způsob využívá více systémových prostředků a spouštění kontejnerů trvá déle.

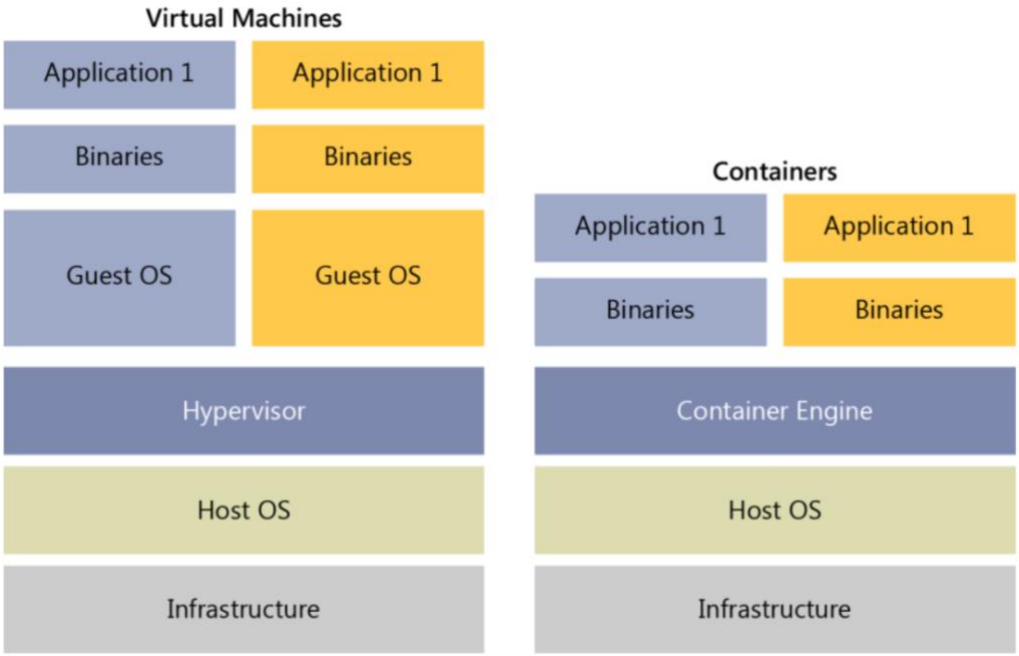
Pro práci s Windows kontejnery je vyžadována instalace Dockeru a tyto kontejnery jsou dostupné pouze v posledních verzích operačních systémů Windows 10, Windows Server 2016 a 2019. Microsoft se zároveň stal zakládajícím členem organizace Open Container Initiative (OCI), která se zabývá vývojem a standardizací kontejnerů. [1][12][13]

4.2 Rozdíl mezi kontejnery a virtuálními stroji

V mnoha ohledech mohou být kontejnery považovány za virtuální stroje (VM – Virtual Machine). Největší rozdíl mezi kontejnery a virtuálními stroji spočívá v tom, že každý virtuální stroj zahrnuje nejen aplikace, ale i celý operační systém. Oproti tomu kontejnery sdílejí jádro hostitelského operačního systému s dalšími kontejnery, z nichž každý běží jako samostatný proces v uživatelském prostoru. Díky tomu využívají kontejnery méně systémových prostředků (operační paměť, procesor, pevný disk), než virtuální stroje.

Z *Obr. 2 Srovnání kontejnerů a virtuálních strojů* vyplývá, že v případě virtuálních strojů je přítomen hypervisor, který umožnuje spuštění několika virtuálních strojů na jednom počítači. Každý virtuální stroj má vlastní jádro, které nesdílí s hostitelským počítačem. Naproti tomu počítač s několika kontejnery spouští jeden operační systém a každý kontejner sdílí jádro hostitelského

operačního systému za pomoci kontejnerizačního softwaru. Představitelem tohoto softwaru může být Docker Engine. [1]



Obr. 2 Srovnání kontejnerů a virtuálních strojů
Zdroj: [13]

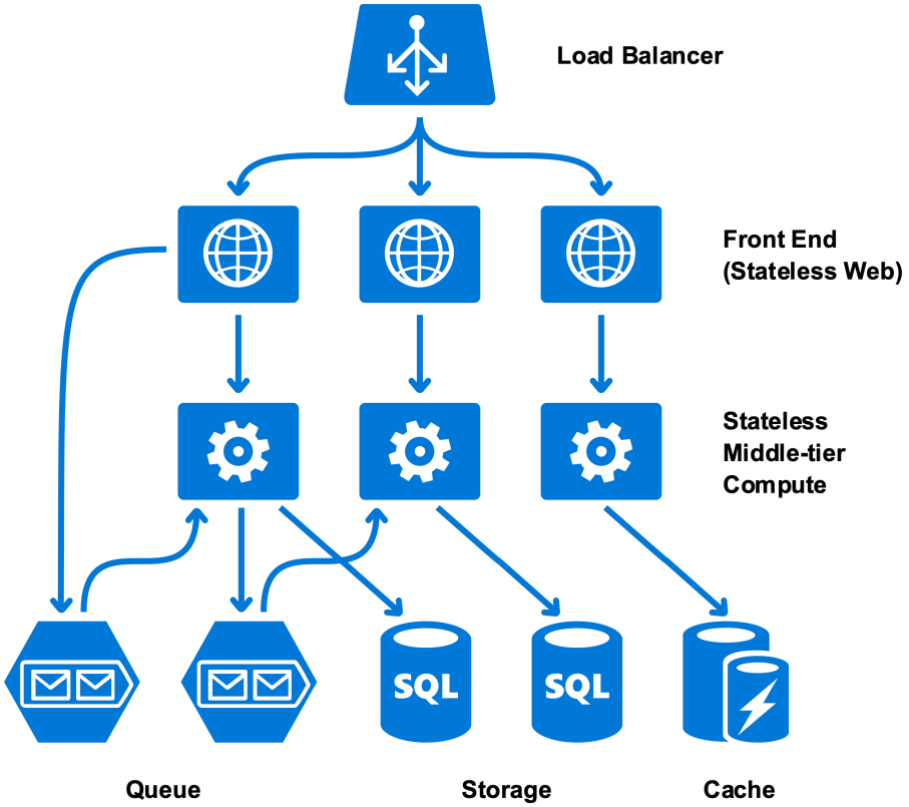
Mezi hlavní výhody kontejnerů oproti virtuálním počítačům patří malá velikost. Kontejnery sdílejí systémové jádro hostitelského operačního systému, a proto nevyžadují operační systém pro každý kontejner, což zvyšuje efektivitu serverů a snižuje náklady na provoz a licence. Z tohoto důvodu se velikost kontejnerů pohybuje v rozmezí pouhých několika megabajtů, zatímco virtuální stroje s vlastním operačním systémem mohou mít velikost až několika gigabajtů. Proto může jediný počítač hostit více kontejnerů než virtuálních strojů, navíc kontejnery mohou být provozovány i ve virtuálních počítačích. Kontejnerizované aplikace se spouštějí okamžitě, jelikož nemusí nabíhat celý operační systém předtím, než se spustí samotná aplikace uvnitř kontejneru. To umožňuje, aby byly kontejnery okamžitě spuštěny, když je to nutné, a zastaveny, pokud již nejsou potřeba.

Nevýhodou kontejnerů je nižší zabezpečení v porovnání s virtuálními počítači. Důvodem je právě sdílení jádra, kvůli kterému jsou kontejnery navzájem méně izolovány než virtuální stroje. [10][14]

4.3 Kontejnerizované aplikace

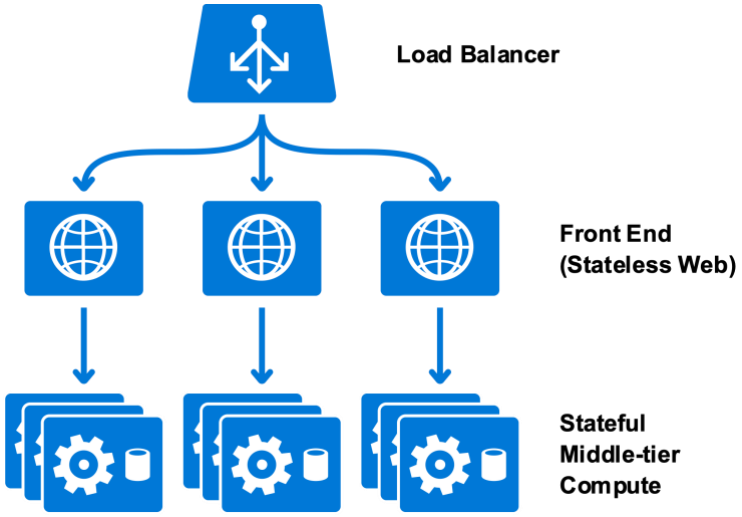
Kontejnery jsou jednoduché, rychle spustitelné, přenositelné, a zároveň mohou být kdykoliv odstraněny. Pokud kontejner selže, nebo se odstraní, dojde současně i ke ztrátě všech dat, která běžící aplikace uložila do souborového systému v kontejneru. V kontejneru by měla být vždy spuštěna pouze jedna aplikace, nikdy by tedy neměl obsahovat například webovou aplikaci a databázi.

Při kontejnerizaci aplikací se doporučuje začít s bezstavovými (stateless) aplikacemi. Jsou navrženy tak, aby lokálně neukládaly žádná data. Všechna data, se kterými aplikace pracuje, by měla být uložena na externím úložišti, do kterého se aplikace připojuje. Kontejnery lze pak jednoduše a rychle spouštět v různých prostředích a škálovat dle potřeby. Příkladem bezstavových aplikací jsou webové aplikace, které ukládají a načítají data z databáze.



Obr. 3 Bezstavové aplikace
Zdroj: [16]

Naproti tomu stavové (stateful) aplikace ukládají data do souborového systému v kontejneru, ve kterém je aplikace spuštěna. Kontejnerizace těchto aplikací je mnohem složitější, pokud v případě selhání kontejneru není žádoucí o data přijít. Aby uložená data zůstala zachována i po odstranění kontejneru, musí být ke kontejneru připojen adresář z hostitelského souborového systému, nebo musí být využita některá ze služeb pro ukládání souborů, jako Amazon S3, EBS volumes nebo OpenStack Swift. Data se pak ukládají mimo kontejner a jsou dostupná i pro nově spuštěné kontejnery. V produkčním prostředí tato práce se soubory není vhodná, jelikož přicházíme o některé výhody kontejnerů, jako například přenositelnost, rychlé spouštění a nezávislost na prostředí. Stavovými aplikacemi jsou typicky databáze.



Obr. 4 Stavové aplikace
Zdroj: [16]

V mnoha případech aplikace vyžadují ke svému spuštění konfigurační informace (adresa a přihlašovací údaje do databází nebo k externím službám), které se mohou pro různá prostředí lišit. Konfigurační informace se předávají do kontejneru pomocí environmentálních proměnných. To znamená, že pokud se kontejner kdykoliv a kdekoliv spustí s nastavenými environmentálními proměnnými, spuštěná aplikace uvnitř kontejneru převezme konfiguraci právě z těchto proměnných. [19]

Bez ohledu na to, jaké typy aplikací jsou spouštěny v kontejnerech, společnosti jako Docker, Kubernetes, Microsoft nebo Amazon poskytují různé nástroje a služby, které umožňují efektivně spravovat a konfigurovat stavové i bezstavové aplikace v kontejnerech.

5 Docker

Docker je open source platforma, licencovaná pod licencí Apache 2.0, určená pro vývojáře a administrátory k vývoji, nasazení a spouštění aplikací v kontejnerech. Docker je napsaný v programovacím jazyce Go a je vyvíjený stejnojmennou společností (dříve dotCloud) ve spolupráci s komunitou uživatelů a společnostmi jako je Microsoft, Rad Hat nebo IBM. [19] V současnosti patří mezi nejoblíbenější, nejpoužívanější a nejžádanější platformy pro správu kontejnerů podle výsledků ročního průzkumu mezi vývojáři, který provedl portál Stack Overflow [18].

Docker nabízí různé nástroje pro správu životního cyklu kontejnerů, které společně s kontejnery poskytují následující výhody [19]:

- **Standardy** – Docker vytvořil standardy pro kontejnery, díky kterým mohou být přenositelné a nasazené kdekoliv. Kontejnery mohou také běžet na fyzickém či virtuálním stroji nebo v cloudu.
- **Bezpečnost a izolovanost** – Docker umožňuje spustit libovolnou aplikaci bezpečně v kontejnerech. Kontejnery jsou od sebe navzájem izolovány a současně lze spustit hned několik kontejnerů na hostitelském počítači.
- **Nízké náklady a malá velikost** – Kontejnery sdílejí jádro hostitelského operačního systému, a proto nevyžadují operační systém pro jednotlivé aplikace. Zvyšuje se tím efektivita serveru a snižují se náklady na systémové prostředky a licence.

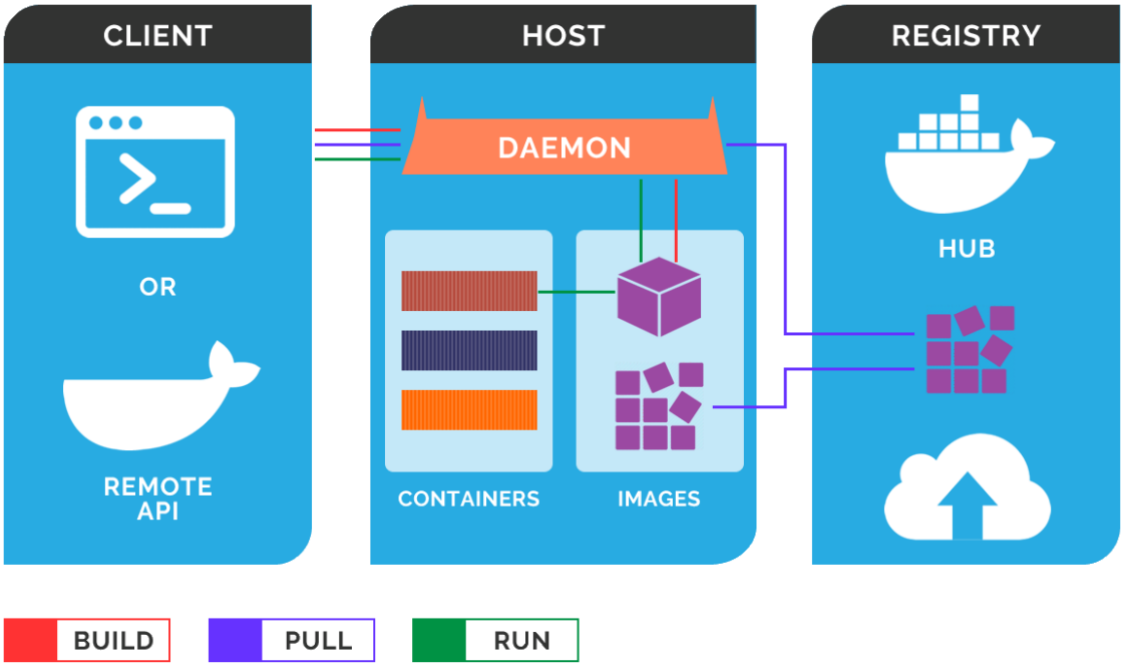
Docker je dostupný ve dvou variantách Community Edition (CE) a Enterprise Edition (EE) a je možné ho provozovat na různých platformách jako je Linux, Windows, Microsoft Azure nebo Amazon EC2. [21]

- **Docker Community Edition** je vhodný pro vývojáře a malé týmy, které chtějí s Dockerem začít, a zároveň experimentovat s aplikacemi běžícími v kontejnerech.
- **Docker Enterprise Edition** je určen zejména pro firmy, které sestavují, provozují a dodávají aplikace v kontejnerech v produkčním prostředí.

Využití nalezne tato technologie v každé fázi vývojového cyklu softwaru a lze ji jednoduše integrovat do procesu continuous integration (CI) a continuous delivery (CD), který umožňuje nepřetržité sestavení, testování a nasazení aplikací do různých prostředí (vývojové, testovací nebo produkční). [20][21]

5.1 Architektura dockeru

Docker je postaven na architektuře klient-server. Uživatel využívá klienta Dockeru ke komunikaci se serverem, tzv. daemonem, který zajišťuje sestavení, provoz a distribuci kontejnerů. Klient i daemon mohou běžet na stejném hostitelském systému, nebo se klient může připojit k daemonu vzdáleně. Klient a daemon komunikují pomocí rozhraní REST API, přes sokety nebo přes síťové rozhraní. [19]



Obr. 5 Architektura Dockeru
Zdroj: [17]

5.1.1 Docker daemon

Docker daemon (dockerd) je služba, která běží na hostitelském operačním systému, naslouchá požadavkům klienta a spravuje Docker komponenty, mezi které patří obrazy, kontejnery, sítě nebo úložiště. Docker daemon může také komunikovat s dalšími daemony pro správu skupiny kontejnerů.

Požadavky přijímá přes tři různé typy socketů *unix*, *tpc* nebo *fd*. Ve výchozím nastavení se využívá *unix* socket nebo v případě systému Windows npipe (named pipe). Obě varianty vyžadují administrátorské oprávnění. *Tcp* socket se používá pro vzdálenou komunikaci, která není ve výchozím nastavení povolena, kvůli bezpečnosti, a vyžaduje další konfiguraci. Systémy, které využívají systemd, používají ke komunikaci s Docker daemonem socket fd (file descriptor). [19]

Docker daemon se automaticky spouští po každém restartování systému s konfigurací, která je definována v souboru *daemon.json*. Konfigurační soubor je ve formátu JSON a udržuje veškerou konfiguraci na jednom místě.

```
{
  "debug": true,
  "tlsverify": true,
  "tlscert": "C:\ProgramData\docker\certs.d\cert.pem",
  "tlskey": "C:\ProgramData\docker\certs.d\key.pem",
  "tlscacert": "C:\ProgramData\docker\certs.d\ca.pem",
  "hosts": ["tcp://192.168.5.220:2376"]
}
```

Zdrojový kód 1 Konfigurační soubor *daemon.json*
Zdroj: Vlastní zpracování

Na ukázce *Zdrojový kód 1 Konfigurační soubor daemon.json* je konfigurace, která spouští Docker daemon v režimu ladění. Navíc je povolena síťová komunikace, která je zabezpečená a Docker deamon naslouchá na adrese 192.168.5.220 na portu 2376.

5.1.2 Docker klient

Docker klient umožňuje uživatelům komunikovat s více než jedním Docker daemonem v infrastruktuře. Ke komunikaci s Docker daemonem poskytuje klient příkazy, které uživatel zapisuje do příkazového řádku (CLI). Mezi nejběžnější příkazy patří:

- **docker run** – Vytvoří a spustí kontejner na pozadí nebo v popředí.
- **docker pull** – Stáhne obraz z registru do Docker daemonu.
- **docker push** – Nahraje obraz z Docker daemonu do registru.
- **docker build** – Sestaví nový obraz kontejneru.
- **docker stats** – Zobrazuje statistiku využití systémových prostředků jednotlivými kontejnery.
- **docker exec** – Spustí příkaz uvnitř kontejneru.
- **docker logs** – Zobrazí protokoly kontejneru.

5.1.3 Registr

Registr je služba, která primárně slouží k ukládání a distribuci již sestavených Docker obrazů. Registr je organizován do repozitářů, ve kterých jsou uchovávány verze konkrétního obrazu. Stejný obraz může mít více různých verzí identifikovaných podle jejich značek (tags). Registr umožnuje uživatelům vyhledat a stáhnout jakýkoliv obraz z repositáře, nebo naopak do něho nahrát a uložit již sestavený obraz, který je následně možné sdílet s ostatními členy vývojového týmu, nebo různými organizacemi. Repozitáře uvnitř registru mohou být nastaveny jako veřejné nebo soukromé.

- **Soukromý** – V repozitáři jsou uloženy obrazy, které mohou obsahovat citlivé informace, nebo zdrojové kódy a jsou sdíleny pouze v rámci týmu nebo organizace.
- **Veřejný** – Obraz z repozitáře je přístupný komukoliv a každý si ho může stáhnout.

Nejběžnější příkazy pro práci s registry jsou *docker pull* a *docker push*. *Zdrojový kód 2 Stažení obrazu z veřejného registru* demonstruje stažení obrazu pojmenovaného *windowsservercore* se značkou *1809* z veřejného registru *logeto.azure.io*.

```
PS C:\> docker pull logeto.azurecr.io/windowsservercore:1809
1809: Pulling from windowsservercore
65014b3c3121: Pull complete
12c8dbabfd62: Pull complete
2a829ee657db: Pull complete
acd0c619ea4b: Pull complete
2ef9c0a457f7: Pull complete
Digest: sha256:41d9705e8421dca6050b257e8cfcd8c4a68c78795bf5a63a2
3b4be5213184328
Status: Downloaded newer image for logeto.azurecr.io/windowsserv
ercore:1809
```

Zdrojový kód 2 Stažení obrazu z veřejného registru
Zdroj: Vlastní zpracování

Společnost Docker provozuje veřejný registr zvaný Docker Hub, který kromě veřejných a soukromých repozitářů poskytuje také automatické sestavení obrazu. Docker Hub obsahuje desítky tisíc obrazů od různých uživatelů a společností. Docker je ve výchozím nastavení nakonfigurovaný tak, aby primárně vyhledával obrazy z něho. Na trhu existuje několik dalších služeb od společností jako Microsoft, Amazon nebo Google, které nabízejí své placené veřejné registry s vysokou dostupností, jedná se o služby Azure Container Registry, Amazon Elastic Container Registry nebo Google Container Registry. Mimo veřejných registrů, provozovaných v cloudu, je možné provozovat i soukromé registry na vlastní infrastruktuře. [10][13][19]

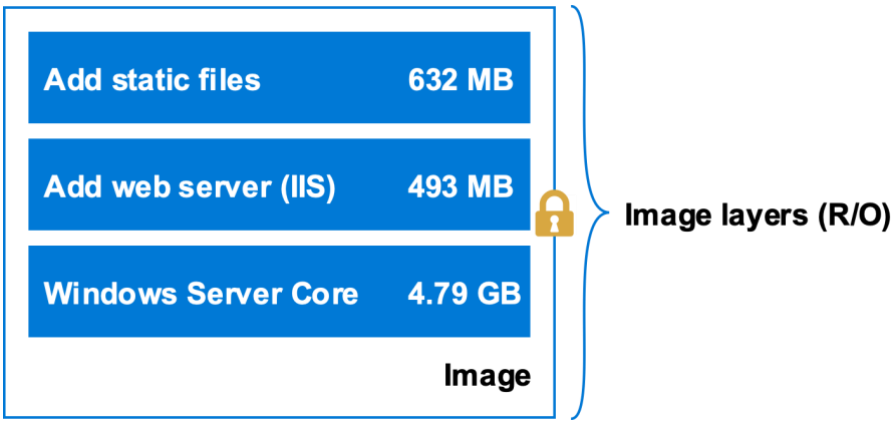
5.2 Docker komponenty

Pro sestavení a spuštění aplikací v kontejnerech Docker vytváří a používá komponenty, mezi které patří obrazy, kontejnery, služby, dockerfiles a další. Následující kapitoly obsahují stručný přehled těchto komponent.

5.2.1 Obraz

Obraz (image) je šablona, ze které lze spustit nové kontejnery. Pro spuštění libovolného počtu kontejnerů lze použít pouze jeden obraz. Obrazy nejsou monolitické bloky, ale skládají se z řady vrstev (layers), které jsou pouze pro čtení. Jednotlivé vrstvy jsou naskládány na sebe a každá vrstva je pouze množinou rozdílů souborového systému od vrstvy před ní. Každý obraz začíná základním obrazem, který tvoří základní vrstvu v obrazu. Typicky je tento základní obraz jedním z oficiálních obrazů, jako je Windows Server Core, Ubuntu nebo CentOS, stažených z veřejných registrů. Je však možné vytvořit obraz zcela od nuly. [13][20]

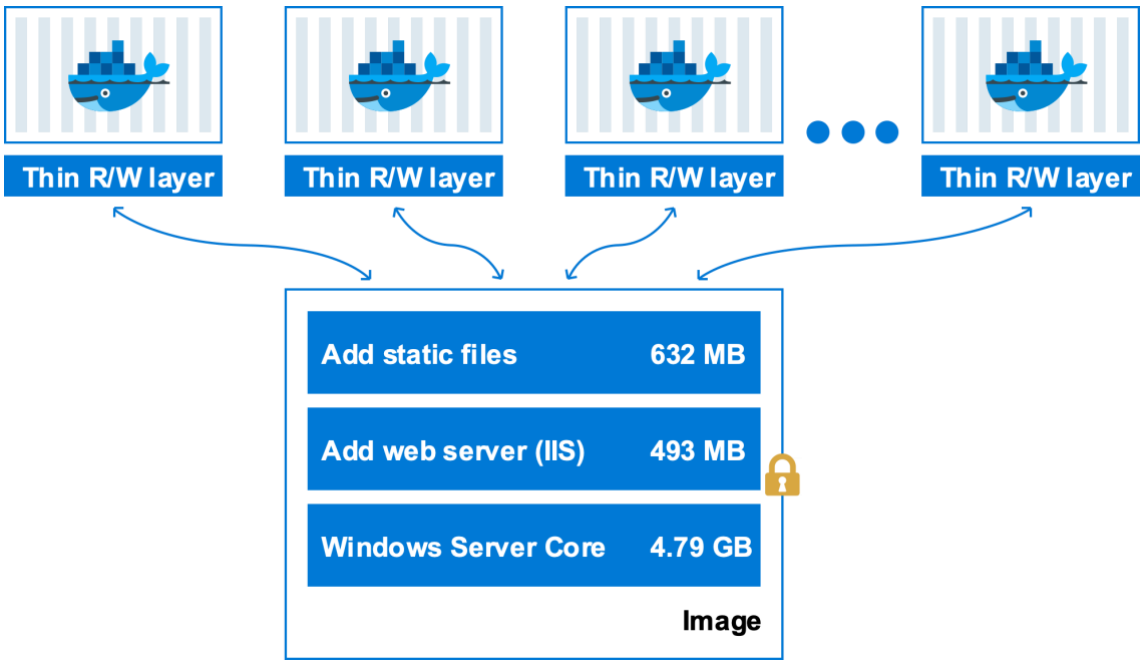
Na *Obr. 6 Obraz* je znázorněno, jak by mohl vypadat vlastní obraz pro webovou aplikaci používající Internet Information Services (IIS) jako webový server.



Obr. 6 Obraz
Zdroj: [20]

Základní vrstva se skládá z Windows Server Core, následuje vrstva, ve které se nainstaluje webový server Internet Information Services. Třetí vrstva obsahuje všechny soubory, které tvoří webovou aplikaci, například HTML, CSS a JavaScript soubory.

Jestliže jsou z obrazu vytvořeny kontejnery, přidá se pro každý kontejner zároveň nová vrstva, do které lze i zapisovat. Tato vrstva se nazývá vrstva kontejneru a jsou v ní uloženy všechny změny provedené ve spuštěném kontejneru (nově vytvořené soubory, změnu existujících souborů, nebo nově smazané soubory). Tyto změny lze uložit do nového obrazu, jinak je možné o změny v kontejneru kdykoliv přijít. Pokud je kontejner odstraněn, smaže se zároveň i vrstva kontejneru a obraz zůstává nezměněn.



Obr. 7 Kontejnery sdílející stejný obraz
Zdroj: [19][20]

Sdílení jednoho obrazu mezi několik kontejnerů má za následek snížení spotřebovaných výpočetních prostředků a dobu spouštění kontejnerů, protože se vytváří pouze tenká vrstva kontejneru a obraz se nemusí znovu vytvářet, nebo stahovat. [19]

5.2.1.1 Sestavení obrazu

Docker obsahuje nástroje, které umožnují sestavit vlastní obrazy podle našich požadavků, nebo lze pouze upravit ty, které vytvořili jiní a publikovali je ve veřejných registrech. Nový obraz je možné sestavit automatizovaným procesem, provedením sady instrukcí obsažených v textovém souboru Dockerfile.

Příklad Dockerfilu (*Zdrojový kód 3 Dockerfile*) má několik řádků, z nichž každý začíná instrukcí jako FROM, RUN nebo CMD. Výsledkem provedení těchto instrukcí je nový obraz, který zahrnuje nakonfigurovaný webový server s aplikací.

```
FROM mcr.microsoft.com/windows/servercore/iis

RUN Add-WindowsFeature Web-Server

WORKDIR /inetpub/wwwroot
COPY content/ .

EXPOSE 80

ENTRYPOINT C:\ServiceMonitor.exe w3svc
```

Zdrojový kód 3 Dockerfile
Zdroj: Vlastní zpracování

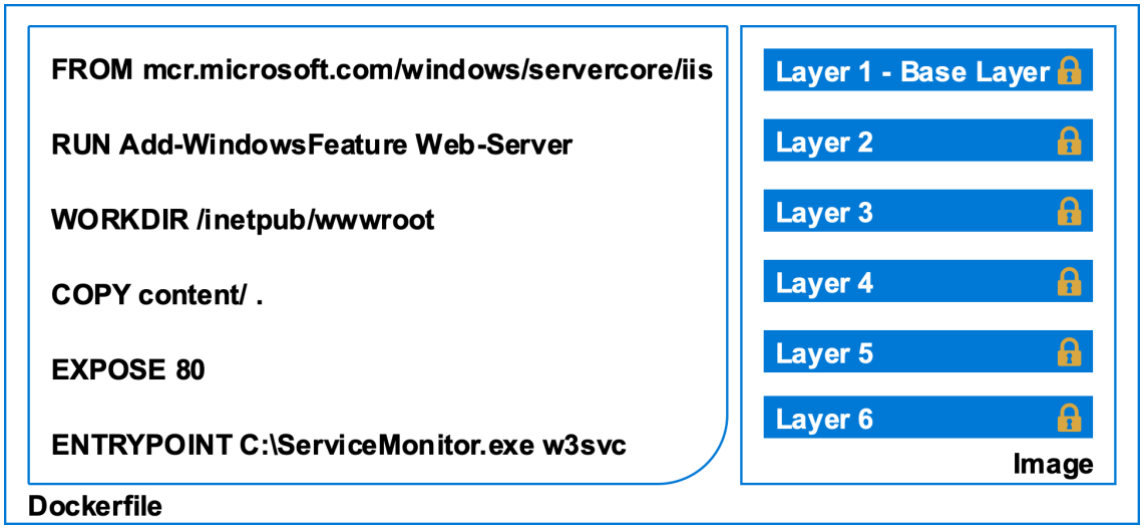
Následující příklady jsou nejčastěji používanými instrukcemi v souborech Dockerfile.

- **FROM** – Většina Dockerfilů začíná právě instrukcí FROM a definuje, ze kterého základního obrazu se vytváří náš vlastní obraz. Při použití instrukce *FROM mcr.microsoft.com/windows/servercore/iis* je výsledný obraz odvozen od základního obrazu operačního systému Windows Server Core a je na něm závislý. Pokud základní obraz není přítomen v systému, kde proces sestavení běží, Docker se pokusí tento obraz stáhnout z veřejného nebo soukromého registru.
- **RUN** – Instrukce RUN provede jakýkoliv platný příkaz a výsledek uloží do nové vrstvy obrazu kontejneru. Tyto příkazy mohou provádět instalaci softwaru, vytváření souborů a adresářů, nebo vytváření konfigurace

prostředí. Dlouhé a složité příkazy lze rozdělit zpětným lomítkem na více řádků. Tím je Dockerfile lépe čitelný, srozumitelný a udržovatelný.

- **COPY a ADD** – Tyto dvě instrukce se používají ke kopírování souborů a složek ze souborového systému hostitele do obrazu, který vytváříme. Instrukce jsou si velmi podobné, s výjimkou toho, že klíčové slovo ADD navíc umožňuje kopírovat sobory ze vzdáleného umístění pomocí URL adresy a kopírovat a rozbalovat TAR soubory.
- **WORKDIR** – Nastavuje pracovní adresář pro další Dockerfile instrukce.
- **CMD a ENTRYPOINT** – Tyto instrukce jsou odlišné od ostatních. Zatímco všechny ostatní instrukce definované v souboru Dockerfile jsou prováděny v době sestavení obrazu, tyto dvě jsou pouze definicí toho, jak a jaký proces, nebo aplikace, bude spuštěna uvnitř kontejneru, když se kontejner spustí z vytvořeného obrazu.

Instrukce jsou prováděny popořadě a každá z nich vytvoří novou vrstvu v obraze, jak je znázorněno na *Obr. 8 Vztah mezi Dockerfile a vrstvami v obraze.* Změní-li se soubor Dockerfile a znovu dojde k sestavení obrazu, budou znovu sestaveny pouze ty vrstvy, které byly změněny. Toto je důvodem, proč jsou obrazy lehké, malé a rychlé ve srovnání s jinými virtualizačními technologiemi.



Obr. 8 Vztah mezi Dockerfile a vrstvami v obraze
Zdroj: Vlastní zpracování

Sestavení obrazu lze spustit příkazem *docker build*, který čte instrukce právě ze souboru Dockerfile. Sestavený obraz lze nahrát do registru, nebo rovnou vytvořit nový kontejner s tímto obrazem. [1][19]

5.2.2 Kontejner

Kontejnery jsou izolovaná prostředí, ve kterých jsou spouštěny aplikace. Kontejner je definován obrazem a dalšími konfiguračními možnostmi. Pomocí rozhraní, které nabízí Docker, lze kontejner spustit, zastavit, přesunout nebo odstranit. Kontejner lze připojit k jedné nebo více sítím, připojit k němu uložiště, nebo vytvořit úplně nový obraz na základě aktuálního stavu. Pokud je kontejner odstraněn, jsou odstraněny i veškeré změny jeho aktuálního stavu, které předtím nebyly uloženy v připojeném úložišti. Kontejnery mají přístup pouze k prostředkům, které jsou definovány v obrazu, pokud nejsou při vytváření kontejneru definovány další přístupy. Vzhledem k tomu, že kontejnery jsou mnohem menší než virtuální stroje, mohou být během několika sekund restartovány a výsledkem je mnohem lepší hustota serverů. [19]

5.2.3 Služba

Služba umožnuje jednoduše škálovat kontejnery se stejnou konfigurací napříč více Docker daemony běžícími na různých hostitelích, které společně fungují jako swarm. Každý člen swarmu je Docker daemon a všichni daemoni spolu úzce spolupracují a komunikují pomocí Docker API. Docker Swarm je více popsán v samostatné kapitole.

Často je služba obrazem mikroslužby v rámci nějaké větší aplikace. Příkladem služby může být webový server, databáze nebo jakýkoliv jiný typ aplikace spustitelné v distribuovaném prostředí. Při vytváření služby je nutné definovat požadovaný stav, například jaký obraz se má použít, jaké porty by měla služba používat, kolik replik kontejnerů by mělo běžet v daném okamžiku, nebo jaké příkazy se mají spustit v běžících kontejnerech. Jednou z klíčových výhod mezi službami a samostatnými kontejnery je to, že lze upravit jejich konfiguraci, aniž bychom museli službu jakkoliv ručně restartovat.[19]

6 Orchestrace kontejnerů

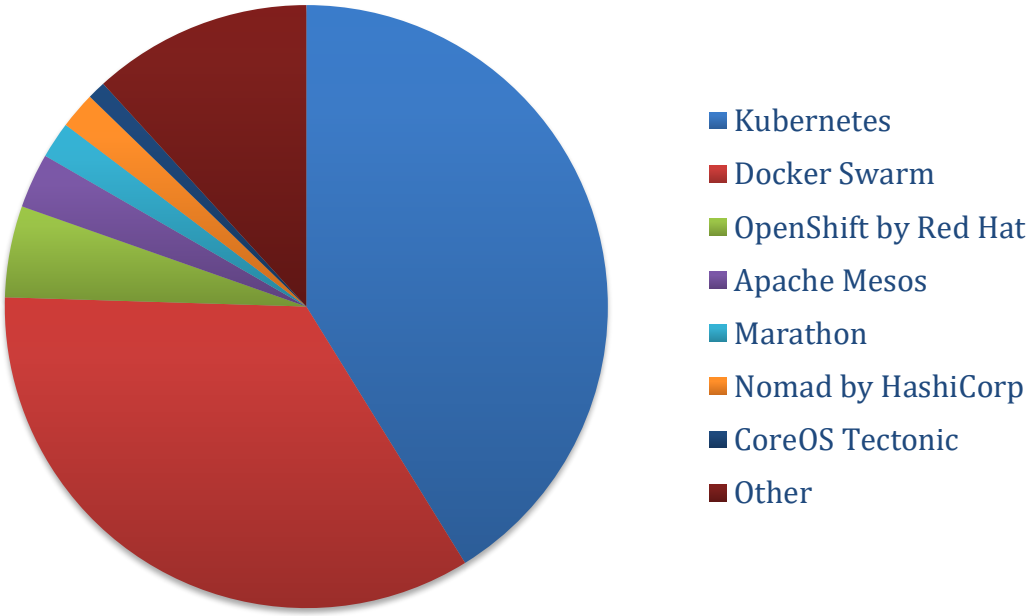
Kontejnery změnily způsob vytváření, nasazování a údržby aplikací. Jestliže je v prostředí spuštěno několik kontejnerů, můžeme tyto kontejnery jednoduše spravovat bez větších potíží na Docker platformě samotné. Pokud se však jedná o podnikové aplikace nebo mikroslužby, kde každá aplikace nebo služba je separována do samostatných kontejnerů, může prostředí obsahovat stovky až tisíce kontejnerů. Z toho důvodu se správa stává mnohem komplikovanější a vyžaduje nějakou formu automatizace.

Pro tyto účely se využívá kontejnerová orchestrace. Jedná se o proces, který automatizuje a spravuje velké množství kontejnerů a jejich vzájemnou interakci ve velkých a dynamických prostředích. V následujícím seznamu jsou uvedeny nejdůležitější operace, které plní nástroje pro orchestraci kontejnerů. [1]

- **Scheduling** – Nástroj pro orchestraci naplánuje nasazení kontejnerů a rozmístí je na nejvhodnější hostitele.
- **Affinity a Anti-affinity** – Určuje, zda se má sada kontejnerů spustit na stejném hostiteli kvůli výkonu, nebo na různých, pokud se jedná o zajištění vysoké dostupnosti.
- **Monitoring** – Sleduje stav kontejnerů a hostitelů.
- **Scaling** – Automaticky přidá nebo odebere instanci kontejneru podle aktuální potřeby.
- **Failover** – Sleduje, co běží na každém hostiteli a v případě poruchy obnoví kontejnery na jiném hostiteli.
- **Update** – Spravuje aktualizace kontejnerů, minimalizuje výpadek aplikace a v případě, že aktualizace selže, je možné vrátit původní stav.
- **Service discovery** – Umožňuje kontejnerům se automaticky lokalizovat, i když jsou spuštěny na různých hostitelských počítačích a mění IP adresy.

Na trhu existuje celá řada open-source nástrojů pro orchestraci kontejnerů, které podporují Docker kontejnery a jejich vývoj kopíruje vznik Dockeru. Podle průzkumu, který prováděla společnost DigitalOcean patří mezi nejpopulárnější

nástroje Docker Swarm a Kubernetes, které budou detailněji popsány v následujících kapitolách [22].



Graf 1 Nejpopulárnější nástroje pro orchestraci kontejnerů
Zdroj: [22]

V závislosti na vybraném nástroji je nezbytné vytvořit konfigurační soubory ve formátu YAML, JSON atd. Tyto konfigurační soubory obsahují veškeré informace (obraz kontejneru, který má být použit, kolik instancí má být spuštěno, které porty mají být otevřené, a další.), které zajistí požadované nasazení kontejnerů nástrojem pro orchestraci. Soubory je vhodné verzovat v některém z verzovacích systémů (Git nebo SVN) a mít tak kdykoliv možnost vrátit se ke starším verzím. Všechny tyto změny konfiguračních souborů se ukládají do repozitářů, ze kterých lze konfiguraci sdílet v různých verzích mezi členy vývojového týmu, a díky tomu lze kontejnery nasadit do různých vývojových a testovacích prostředí ještě před jejich nasazením do produkčního prostředí.

Nástroje pro orchestraci rozdělují jednotlivé kontejnery do logických skupin, které jsou rozmístěny na různé hostitele (fyzické nebo virtuální stroje). Hostitelé jsou předem nakonfigurováni pro spouštění kontejnerů a jsou vzájemně propojení, proto se navenek tváří jako jeden tzv. cluster. V případě nasazení nového kontejneru

do clusteru, nástroj pro orchestraci zajistí naplánování takového počtu kontejnerů, který bude požadován. Pokud obraz kontejneru ještě není k dispozici na cílových hostitelích, kde mají být kontejnery spuštěny, plánovač zajistí, aby byly nejprve staženy z registru. Dále jsou kontejnery spuštěny se všemi nastaveními, které byly definovány v konfiguračních souborech, jakou jsou sítě, ke kterým se mají připojit. nebo porty, které mají vystavit.

Jakmile jsou kontejnery spuštěny, běží v požadovaném stavu, a nástroj pro orchestraci spravuje jejich životní cyklus podle definice z Dockerfilu. Pokaždé, když nástroj pro orchestraci objeví nesoulad mezi skutečným a požadovaným stavem, snaží se co nejlépe obnovit požadovaný stav. Rozpor mezi stavy může vzniknout v případě, kdy nástroj pro orchestraci zjistí, že kontejner obsahuje starou verzi obrazu, a nebo se liší počet požadovaných instancí kontejnerů.

6.1 Docker Compose

Docker Compose lze považovat za základní a jednoduchý nástroj pro orchestraci jednoho i více kontejnerů na jednom hostiteli vyvíjený společností Docker. Využívá strukturovaný konfigurační soubor YAML, ve kterém je uložena definice jedné nebo více kontejnerizovaných aplikací. Pro každou kontejnerizovanou aplikaci (databáze, webový server, fronta atd.), je možné následně vytvořit a spustit kontejner se všemi závislými komponentami, jako jsou sítě nebo sdílené svazky. To vše lze pouze jedním příkazem, který nástroj poskytuje. Konfigurační soubory lze dále doplnit o proměnné prostředí, nebo je různě rozšiřovat a slučovat. To umožňuje libovolně přizpůsobit konfiguraci pro různá prostředí nebo pro různé uživatele.

```yaml
version: '3'
services:
  web:
    image: logeto.azurecr.io/web:8.0.3.2043_1809
    container_name: web
    restart: always
    build:
      context: ./web
      args:
        - VERSION=7.9.2.1569_1809
        - WINDOWS_VERSION=1809
    environment:
      SETTING_DebugMode: "true"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - "C:/Containers Data/Logs:C:/Logs"
```

Zdrojový kód 4 Docker Compose
Zdroj: Vlastní zpracování

Konfigurační soubor se skládá z několika vrstev a objektů, které jsou odděleny pomocí mezer nebo tabulátorů. Každý soubor obsahuje dva objekty na nejvyšší úrovni *version* a *services*. Version určuje formát souboru a podporované objekty. Services obsahuje definici kontejnerizovaných aplikací. V tomto případě se jedná o službu pojmenovanou *web*, která spustí jeden kontejner s webovou aplikací. Služba specifikuje název a obraz kontejneru, který se Docker Compose pokusí stáhnout z veřejného registru. Pokud obraz ve veřejném registru neexistuje, bude automaticky sestaven na základě definice v objektu *build*. Kontejner je spuštěn s environmentální proměnnou, která spouští webovou aplikaci v režimu ladění. Webová aplikace je dostupná na portu 80 a 443 a ukládá své soubory do souborového systému hostitele, takže v případě selhání kontejneru nebudou nenávratně odstraněny.

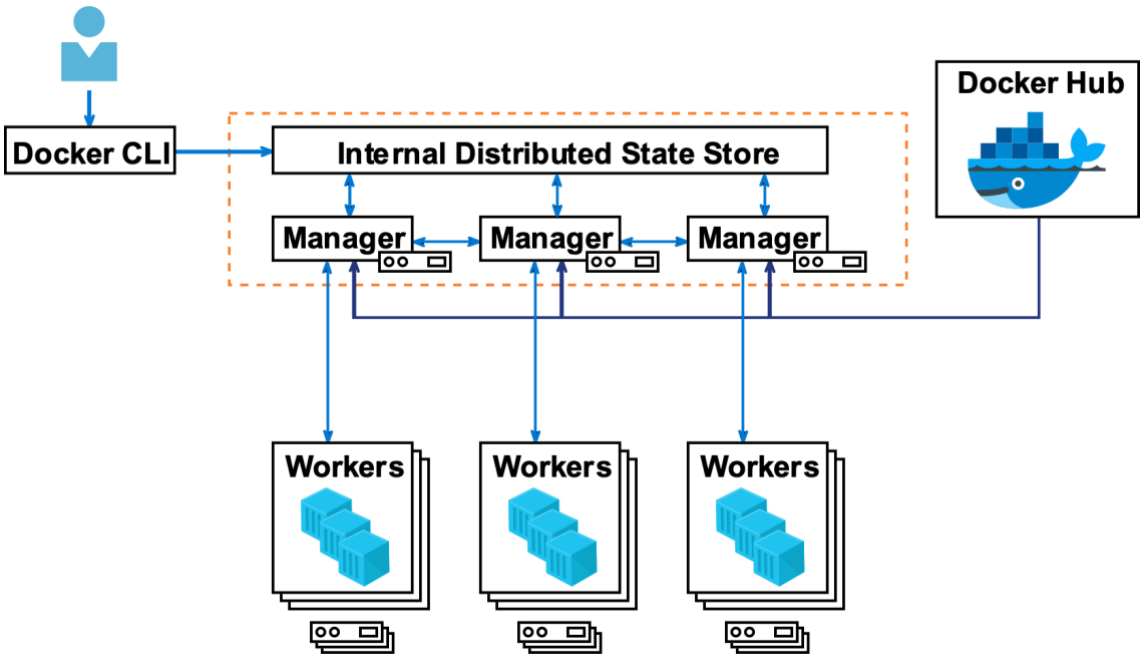
Docker Compose převážně využívají vývojáři nebo testeři, kteří si mohou jednoduše připravit prostředí pro vývoj a testování s předem nadefinovanými službami. Díky své jednoduchosti má nástroj řadu nevýhod. Nepodporuje škálování, load balancing a ve výchozím nastavení spouští všechny kontejnery pouze na jednom hostiteli. Z toho důvodu není zajištěna vlastnost vysoké dostupnosti,

která je pro produkční prostředí zásadní. Existuje však nástroj Docker Swarm zaměřený právě na produkční prostředí, který některé nedostatky řeší.

6.2 Docker Swarm

Docker Swarm je pokročilý nástroj pro správu clusteru a orchestraci Linux i Windows kontejnerů, který je součástí instalačního balíčku Docker od verze 1.12. Umožňuje jednoduše nasazovat a organizovat velký počet kontejnerů na více hostitelích. Navíc podporuje load balancing, monitoring, škálování atd. Díky těmto vlastnostem je zajištěna vysoká dostupnost a lze ho na rozdíl od Docker Compose použít i v produkčním prostředí.

Swarm se skládá z jednoho nebo více uzlů (nodes), viz. *Obr. 9* . Uzle představují fyzické nebo virtuální počítače, na kterých je spuštěn Docker daemon v režimu swarm. Jednotlivé uzle mohou být buď pracovní (workers), manažerské (managers), nebo mohou vykonávat obě role zároveň.



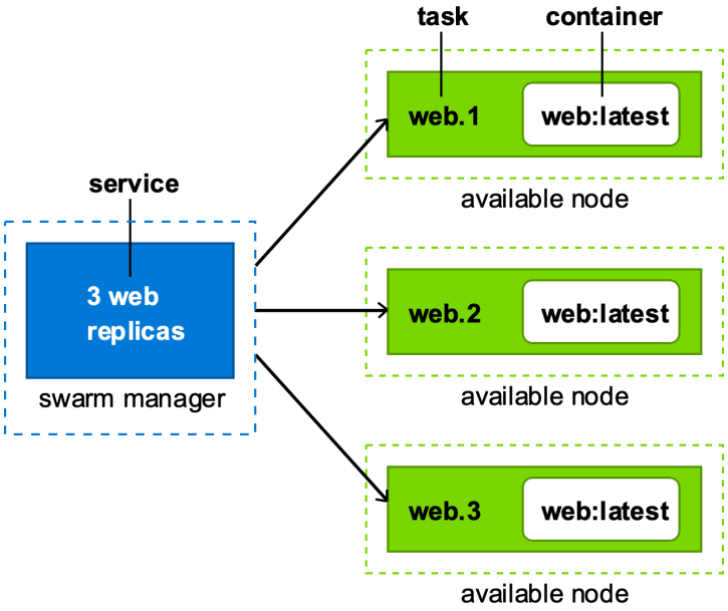
Obr. 9 Architektura Docker Swarm
Zdroj: [23]

Pracovní uzle přijímají a provádějí požadavky od manažerských uzlů a nemají povědomí o celkovém stavu clusteru. Jejich jediným účelem je spouštění kontejnerů. Každý uzel obsahuje agenta, který pravidelně zasílá informace o svém stavu manažerovi. Manažerské uzle přijímají definice služeb od uživatelů a jejich povinností je zajistit konzistentní stav clusteru a služeb, které na něm běží, pomocí implementace algoritmu Raft. Swarm může obsahovat více manažerských uzlů, ale vždy v něm musí být alespoň jeden. Uživatelé mohou komunikovat pouze s manažerskými uzly a spouštět v nich příkazy, které nabízí rozhraní Docker CLI.

Místo definování jednotlivých kontejnerů se v Docker Swarm definují služby, ve kterých je možné nastavit počet replik kontejnerů, nebo způsob aktualizace. Služba je definicí úloh, které jsou spuštěny na manažerských nebo pracovních uzlech. Tato úloha vytvoří kontejner a provede v něm příslušné příkazy. Služby se rozdělují na globální a replikované. Pokud je služba globální, bude spuštěn kontejner na každém dostupném pracovním uzlu v clusteru. U replikovaných služeb distribuuje manažer určitý počet kontejnerů mezi uzle na základě požadovaného stavu z definice. Swarm ve výchozím nastavení používá load balancing ke zveřejnění běžících služeb a k distribuci požadavků mezi službami na základě doménového jména. Jednotlivé služby lze nasadit do swarmu pomocí příkazu *docker service create* nebo *docker stack deploy*, pokud je definice služeb uložená v konfiguračním souboru YAML.

```
docker service create \
  --name web \
  --replicas 3 \
  logeto.azurecr.io/web:8.0.3.2043_1809
```

Na základě příkazu přijme manažer definici služby jako požadovaný stav pro službu. Poté naplánuje na třech pracovních uzlech úlohy, z nichž každá spustí kontejner. Zároveň je mezi kontejnery prováděn load balancing.



Obr. 10 Služba, úloha a kontejner v Docker Swarm
Zdroj: [19]

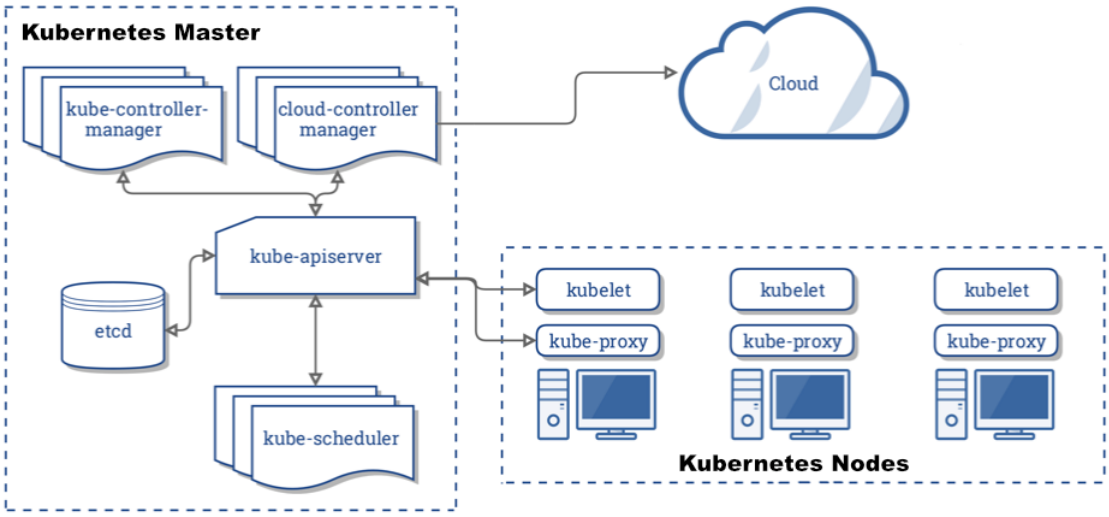
Výhodou tohoto nástroje pro orchestraci kontejnerů je jeho jednoduchá instalace a integrace s jinými nástroji, které nabízí platforma Docker. Stejným způsobem, jakým lze definovat kontejnery v Docker Compose, lze definovat požadovaný stav služeb v Docker Swarmu. Vývojáři a správci nemusí proto vytvářet zcela nové konfigurační soubory a učit se novou strukturu, ale stačí pouze rozšířit ty stávající.

6.3 Kubernetes

Kubernetes (zkráceně k8s) je přenositelná a rozšiřitelná open-source platforma, jejíž hlavním cílem je odstranit složitost správy velkého množství kontejnerů. Nabízí rozhraní API, přes které lze popsat požadovaný stav aplikace a platforma zajistí vše nezbytné pro přizpůsobení infrastruktury. Nasadí skupinu kontejnerů, replikuje je a v případě selhání některých z nich je znovu automaticky obnoví. Kubernetes byl původně navržen společností Google pro masivní škálování. Návrh vychází ze získaných zkušeností s interním nástrojem pro orchestraci kontejnerů Borg. V roce 2014 byl darován organizaci Cloud Native Computing Foundation (CNCF), která spolu s početnou uživatelskou komunitou stojí za popularitou této platformy. Nové funkce jsou vydávány přibližně každé tři měsíce a z toho je většina implementována právě na základě poptávky komunity.

6.3.1 Architektura Kubernetes

Jelikož je Kubernetes open source s relativně malými omezeními, může běžet téměř kdekoliv. Na fyzických nebo virtuálních strojích, případně ve veřejném nebo soukromém cloudu, jako je Microsoft Azure, Google Cloud nebo Amazon Web Services. Nejčastěji se používá společně s Dockerem, který poskytuje standard pro vytváření a distribuci kontejnerů. Kubernetes podporuje ale i další kontejnerové technologie, které splňují standardy Open Container Initiative (OCI). Jednou z nevýhod pro začínající uživatele je složitá konfigurace.



Obr. 11 Architektura Kubernetes
Zdroj: [24]

Kubernetes tvoří alespoň jeden řídící a více pracovních uzlů. Řídící (master) uzel je vstupní bod pro všechny administrativní úkoly a koordinuje všechny aktivity v clusteru, jakou jsou udržení požadovaného stavu nebo nasazení, škálování a aktualizace kontejnerů. Pro zajištění vysoké dostupnosti může být v clusteru více, než jeden řídící uzel. Řídící uzel obsahuje následující základní komponenty:

- **Kube-scheduler** – Komponenta je zodpovědná za rozmístění podů (skupina jednoho nebo více kontejnerů) na různé uzle. Rozmístění závisí na řadě faktorů, jako jsou požadavky na systémové prostředky, software nebo umístění dat. Pokud například aplikace vyžaduje 1 GB operační paměti a 2 jádra CPU, budou pody pro tuto aplikaci rozmístěny na uzly, které mají dostatek prostředků.
- **Etcd** – Distribuované úložiště klíč-hodnota, které Kubernetes používá k ukládání konfigurace clusteru (jako je počet podů, jejich stav, jmenné prostory atd.). Etcd komunikuje pouze přímo s kube-apiserver, který slouží jako zprostředkovatel a validátor pro jakékoliv operace s datovým úložištěm. Všechny ostatní komponenty, které potřebují získat informace o aktuálním stavu clusteru, musí komunikovat skrze kube-apiserver.
- **Kube-apiserver** – Kubernetes API je centrální řídící komponenta, která přijímá všechny REST požadavky a slouží jako frontend do clusteru. Z toho

důvodu slouží jako rozhraní pro veškerou komunikaci v clusteru. Uživatelé mohou komunikovat se serverem API přes příkazový řádek, prostřednictvím nástroje *kubectl*. Pomocí příkazu kubectl lze vytvářet, editovat a mazat jednotlivé prostředky, zobrazovat přehled o aktuálně běžících prostředcích a mnoho dalšího.

- **Kube-controller-manager** – Spouští na pozadí řadu různých procesů, které sledují aktuální stav clusteru a provádí různé operace, aby zajistily požadovaný stav. V případě, že nastane změna konfigurace (například změna obrazu, ze kterého jsou pody spuštěny nebo změna parametrů v konfiguračním souboru yaml), controller zaznamená změnu a aktualizuje všechny prostředky, aby odpovídala novému požadovanému stavu.
- **Cloud-controller-manager** – Komponenta je integrována do každého veřejného cloudu pro optimální podporu zón dostupnosti, virtuálních strojů, síťových služeb, úložiště, směrování a load balancingu.

Pracovní uzel je virtuální nebo fyzický stroj, který spouští kontejnery a je spravován řídícím uzlem. Každý uzel musí mít nainstalovaný nástroj pro správu a provoz kontejnerů, například Docker nebo Rkt. Původně bylo možné pracovní uzly konfigurovat pouze v systému Linux, ale od verze 1.14 je přidána podpora i pro Windows Server 2019. V produkčním prostředí by měl mít cluster minimálně tři uzly. Níže jsou popsány hlavní komponenty v pracovním uzlu.

- **Kubelet** – Hlavní služba v uzlu, která pravidelně přijímá nové nebo upravené specifikace podů a zajišťuje, že pody a jejich kontejnery jsou bez chyb a běží v požadovaném stavu. Specifikace podu jsou soubory psané v YAML nebo JSON. Tato komponenta také zasílá informace řídícímu uzlu o stavu hostitele, na kterém služba běží.
- **Kube-proxy** – Komponenta, která běží na každém pracovním uzlu za účelem řešení jednotlivých podsítí hostitele a vystavení služeb vnějšímu světu. Předává požadavky správným kontejnerům napříč různými izolovanými sítěmi v clusteru.

6.3.2 Resources

Pro seznámení se základní architekturou Kubernetes je důležité znát a pochopit i další prostředky, které umožňují v clusteru provozovat kontejnery tím nejlepším možným způsobem. Mezi nejpoužívanější prostředky (tzv. resources) patří Pods, Services, Deployments, Namespaces, Secrets, ConfigMaps atd.

Pod je nejzákladnějším prostředkem v Kubernetes, který lze vytvořit nebo spravovat. Každý pod představuje jednu instanci dané aplikace nebo spuštěného procesu v Kubernetes. Skládá se z jednoho nebo více na sobě závislých kontejnerů, které sdílejí stejné síťové zdroje, úložiště, jedinečnou síťovou IP adresu a další konfiguraci, jak tyto kontejnery budou spuštěny. Kontejnery uvnitř podu mezi sebou komunikují pomocí portů a je možné do nich připojit datové úložiště s daty. Pody jsou vytvářeny a ničeny na uzlech podle potřeby, aby odpovídaly požadovanému stavu určeného uživatelem ve specifikaci podu. V případě selhání nebo nedostatku prostředků na uzlu je běžící pod na tomto uzlu zničen. Z toho důvodu je mnohem běžnější spravovat pody pomocí controllerů, které umožňují vytvářet a spravovat více podů, škálovat je, aktualizovat, nebo je automaticky obnovit po selhání na jiném uzlu. Mezi příklady controllerů, které obsahují jeden nebo více podů patří:

- **ReplicaSet** – Zajišťuje spuštění zadaného počtu podů. Využívá se k zajištění vysoké dostupnosti a odolnosti proti chybám.
- **Deployment** – Deployment spravuje controller ReplicaSet, který řídí počet podů, které by měly být spuštěny na základě požadovaného stavu. Navíc spravuje několik dalších věcí, jako jsou aktualizace podů a vrácení změn do původního stavu v případě, že aktualizace selže.
- **StatefulSet** – Je přizpůsoben pro správu podů, které vyžadují perzistentní úložiště. Využívají se pro stavové aplikace jako je MySQL, Elasticsearch nebo MongoDB.
- **DaemonSet** – Je způsob, jak zajistit, aby na každém uzlu v clusteru běžela jedna instance podu. Používá se pro služby, které zajišťují monitorování, protokolování a skenování virů.

- **Job a CronJob** – Spouštějí naplánovanou úlohu, která může pravidelně spouštět nějaké zálohovací operace nebo import dat z jiného systému.

Jelikož pody v clusteru jsou vytvářeny podle potřeby a mohou kdykoliv zaniknout, není zaručeno, že jejich IP adresa zůstane stejná. Proto nemá smysl používat jejich IP adresu. Služby (services) v Kubernetes seskupují pody a zajišťují mezi nimi komunikaci uvnitř clusteru, zároveň je umožňuje zpřístupnit mimo cluster do veřejné sítě. Služby poskytují stabilní IP adresu, která trvá po celou dobu existence služby, i když se změní IP adresy členských podů. Klienti zasílají požadavky na jednu stabilní IP adresu a jejich požadavky jsou přesměrovány na jeden z podů v rámci služby. Služba identifikuje své pody pomocí štítků a selektorů. Aby mohl být pod členem služby, musí mít všechny své štítky uvedené v selektoru služby. Štítek tvoří pár klíč-hodnota, který slouží k popisu a propojení objektů. [25] Existuje několik typů služeb, které určují, jak je služba vystavena do sítě a odkud je služba přístupná.

- **ClusterIP** – Výchozí typ služby, který vystavuje službu na interní IP adrese clusteru. Služba je dostupná pouze v rámci clusteru.
- **NodePort** – Vystavuje službu na IP adrese každého uzlu na konkrétním portu. Služba může zpracovávat požadavky, které přicházejí z vnějšku clusteru.
- **LoadBalancer** – Služba je přístupná z vnějšku clusteru prostřednictvím loadbalanceru od poskytovatele cloudu. Poskytovatel cloudu zajistí vytvoření loadbalanceru, který pak automaticky směruje požadavky na službu v Kubernetes.
- **ExternalName** – Mapují službu na název DNS. Není stanoveno žádné proxy jakéhokoliv druhu. Toto se běžně používá k vytvoření služby v Kubernetes, která představuje externí datové uložiště, jako je databáze, která běží mimo Kubernetes.

S přibývajícími službami se správa může stát velmi složitá. Kubernetes proto podporuje prostředek Ingress, který funguje jako vstupní bod do clusteru. Umožňuje sloučit směrovací pravidla do jednoho zdroje, proto může vystavit více

služeb pod stejnou IP adresou. Navíc podporuje loadbalancing, TLS akceleraci a řízení certifikátů. Typickou implementací Ingress je Treafik, Ambassador nebo NGINX. [8]

Doposud byly popsány hlavní prostředky, bez kterých by provoz kontejnerů v Kubernetes nebyl možný. Mezi další podpůrné prostředky, se kterými je možné se v této práci setkat jsou:

- **Namespace** – Jeden fyzický cluster může spouštět více virtuálních clusterů. Virtuální cluster je určený pro prostředí s mnoha uživateli rozloženými do více týmů nebo projektů. Využívá se pro oddělení produkčního a vývojového prostředí.
- **Volume** – Datový svazek se vztahuje na celý pod a je připojen do všech kontejnerů v podu. Kubernetes zaručuje, že data jsou zachována i po restartu kontejnerů. Jeden pod může mít připojeno více svazků různých typů.
- **Secret** – Obsahuje citlivé informace, jako jsou hesla, certifikáty TLS, tokeny OAuth a klíče ssh.
- **ConfigMap** – Je mechanismus, jak do kontejnerů připojit konfigurační soubory.

Tyto prostředky jsou definovány pomocí konfiguračních souborů (tzv. manifestů) ve formátu YAML nebo JSON. [26] Kubernetes nepodporuje konfigurační soubory napsané pro Docker Compose nebo Swarm, jelikož mají zcela odlišné objekty a strukturu. Konfigurační soubory je možné porovnat na ukázkách *Zdrojový kód 5 Kubernetes manifest* a *Zdrojový kód 4 Docker Compose*. Obě konfigurace definují webovou aplikaci se stejnou konfigurací.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: logeto
  labels:
    app: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      name: logeto
      labels:
        app: web
    spec:
      nodeSelector:
        kubernetes.io/os: windows
      imagePullSecrets:
      - name: logeto-acr
      containers:
      - name: logeto
        image: logeto.azurecr.io/web:8.0.3.2043_1809
        ports:
        - name: http
          containerPort: 80
        - name: https
          containerPort: 443
        env:
        - name: SETTING_DebugMode
          value: "true"
        volumeMounts:
        - mountPath: "C:/Logs"
          name: logs
      volumes:
      - name: logs
        hostPath:
          path: "C:/Containers Data/Logs"
```

Zdrojový kód 5 Kubernetes manifest
Zdroj: Vlastní zpracování

II. Praktická část

7 Aplikace Logeto

Aplikace Logeto (na českém a slovenském trhu známá pod názvem Výkaz práce) je systém pro jednotlivce i velké firmy. Slouží k jednoduché evidenci práce a docházky na pracovišti, na služebních cestách nebo z domova. Aplikace je postavena na platformách ASP.NET, Xamarin, Universal Windows Platform a Microsoft SQL server. Vyvíjena je společností Systemart s.r.o., která byla založena v roce 2003, se sídlem v Hradci Králové a dalšími pobočkami v Praze a Brně.

7.1 Popis aplikace

Aplikace je rozdělena do několika agend, které slouží k evidenci a vyhodnocení zadaných údajů. Uživatelé mají k dispozici široké možnosti nastavení aplikace, které vyhoví i náročným uživatelům. Následující agendy pokrývají veškerou funkcionalitu aplikace.

- **Docházka –** Evidence docházky dělá z aplikace plnohodnotný docházkový systém. Docházka umožňuje pracovníkům snadno zaznamenávat příchody, odchody, absence nebo přestávky na pracovišti. Současně se zaznamenáváním práce dochází k průběžnému ověřování lokality, kde se pracovník při práci aktuálně nachází. Díky tomu lze získat přehled o přítomnosti pracovníků na pracovišti a věrohodné podklady pro mzdy. Záznamy docházky jsou především pro podporu hlavní agendy Výkaz práce, ale lze je zobrazit i samostatně.
- **Výkaz práce –** Hlavní agenda Výkaz práce slouží jako evidence odpracovaných hodin jednotlivých pracovníků s možností rozepisování na jednotlivé zakázky a jejich části. Spolu se zakázkami lze zaznamenávat i další požadované údaje (fakturovatelnost, nákladové a fakturační sazby, popis nebo uživatelsky nakonfigurované vlastní pole). Výstupem jsou přehledné exporty a tiskové sestavy, které poskytují přehledné pracovní výkazy a podklady pro výpočet mzdy.

- **Kniha jízd –** Pomocí integrované knihy jízd aplikace umožňuje evidovat jednotlivé jízdy vozidel. Jízdy lze rozepsat na zakázky a pomocí nastavených sazeb za vozidlo lze sledovat i příslušné náklady. Zajímavou funkcí při vykazování jednotlivých jízd je možnost automatického generování knihy jízd přímo z Docházky nebo Výkazu práce, na základě průjezdných bodů a navštívených lokalit.
- **Výdaje –** Agenda nabízí možnost evidovat jednorázové výdaje nebo výnosy, které svým charakterem nespadají do agend Kniha jízd nebo Výkaz práce. Jednotlivé výdaje a výnosy lze rozšířit o větší či menší množství podrobností a mít tak přehled uzpůsobený vlastním potřebám.
- **Zakázky –** Veškerou vykázanou činnost pracovníků a jízdy vozidel lze evidovat na jednotlivé zakázky, popř. jejich členění. U jednotlivých zakázek a jejich členění je možné navíc nastavovat časový a nákladový rozpočet, nebo je pomocí oprávnění zpřístupnit pouze určité skupině pracovníků. Agenda nabízí přehledné tiskové sestavy a exporty, které poskytují ucelený přehled o jejich ziskovosti.
- **Fakturace –** Agenda je určena pro snadné a rychlé vytváření podkladů pro fakturaci z vykázaných jízd vozidel nebo činnosti pracovníků. Na základě nastavených fakturačních sazeb a vybraných záznamů k fakturaci aplikace automaticky vytvoří podklad pro fakturaci s odpovídající celkovou částkou k úhradě.

Aplikace Logeto je vyvíjena jako webová aplikace, ke které uživatelé přistupují prostřednictvím webového prohlížeče, přičemž jsou aktuálně podporovány Microsoft Edge, Google Chrome, Firefox a Safari. Současně s webovou aplikací probíhá intenzivní vývoj aplikací pro mobilní telefony a osobní počítače. Pro vývojáře třetích stran je k dispozici veřejné rozhraní REST API. Využívá se k propojení, integraci a synchronizaci s dalšími externími systémy. Obvykle se jedná o interní systémy zákazníka.

Celý systém si lze pro jednoduchost představit jako architekturu klient-server. Server zde zastupuje právě webová aplikace, která řídí připojené klienty a vyhodnocuje veškeré záznamy. Osobní počítače, terminály a mobilní zařízení jsou

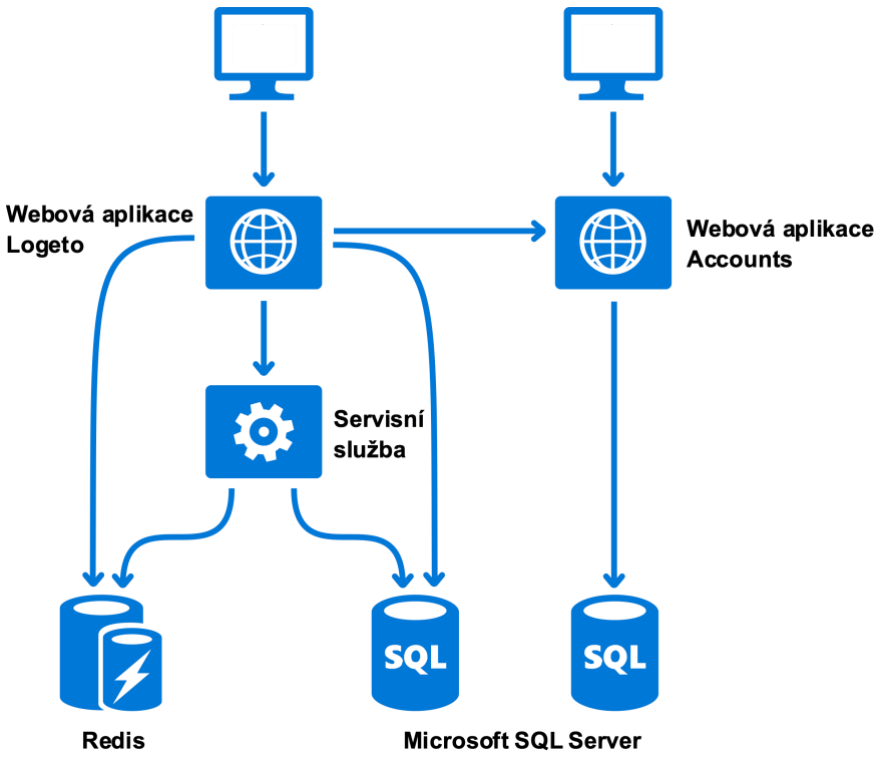
klienti, kteří se převážně starají o sběr dat od uživatelů. Jednotlivá zařízení se neustále na pozadí synchronizují, takže všichni uživatelé mají k dispozici vždy aktuální data.

- **Webová aplikace** – Webová část aplikace je základním kamenem celého systému, bez kterého nelze aplikaci Logeto efektivně používat. Nachází se zde hlavní úložiště dat a veškeré nastavení systému. Poskytuje oproti ostatním aplikacím přehledné tiskové sestavy, přehledy i možnosti exportu dat do formátu Excel nebo PDF.
- **Mobilní aplikace –** Jedná se o velice využívanou aplikaci, která vyniká svojí jednoduchostí. Je vhodná pro všechny uživatele, kteří mají při práci k dispozici mobilní telefon s operačním systémem iOS (11 a vyšší) a Android (4.0.3 a vyšší). Spolu se zadáním práce a docházky je zaznamenávána i aktuální poloha pracovníka, díky které lze mít přehled o pohybu pracovníků i o čase stráveném na jednotlivých lokalitách.
- **Aplikace pro osobní počítače –** Klientská aplikace určená pro osobní počítače s operačním systémem Windows 10, která je rychlá a snadno ovladatelná. Od mobilních aplikací se liší tím, že v ní lze přepínat i na jiné pracovníky, u kterých má přihlášený uživatel oprávnění zakládat záznamy, nebo do nich nahlížet. Aplikaci je možné používat i v případě nedostupného internetového připojení. Po opětovném obnovení internetového připojení se data opět synchronizují s ostatními zařízeními.
- **Aplikace pro terminál –** Poslední klientská aplikace funguje jako moderní docházkový terminál a je vhodná především pro trvalá pracoviště. Aplikaci lze nainstalovat na libovolné zařízení nejlépe s dotykovou obrazovkou a s operačním systémem Windows 10. Z důvodu bezpečnosti je aplikace spuštěna ve výhradním režimu (Assigned access), ve kterém pracovníci nemohou zařízení používat pro jiný účel než pro zaznamenávání práce. Jednotliví pracovníci se identifikují NFC (Near field communication) čipem, přiděleným kódem, nebo otiskem prstu. Firma Systemart s.r.o. vyvíjí a prodává své vlastní zařízení včetně modulárních doplňků v podobě různých čteček.

Aplikace Logeto je dostupná zcela bezplatně, pokud je využívána pouze jedním aktivním pracovníkem. V případě větších společností s více pracovníky je aplikace po dvouměsíčním bezplatném období zpoplatněna dle aktuálního platného ceníku. Vyúčtování je prováděno na začátku každého kalendářního měsíce. Za každého aktivního pracovníka, který zadal záznamy do agendy Výkaz práce nebo Kniha Jízd je odečtena příslušná částka z kreditu. Kredit lze pohodlně dobíjet různými platebními metodami přímo ve webové aplikaci, kde se také nachází přehled jednotlivých vyúčtování a plateb. [27][28][29]

7.2 Architektura aplikace

Tato práce se zaměřuje hlavně na kontejnerizaci již existující webové aplikace. Následující diagram ilustruje architekturu aplikace, včetně veškerých komponent, které aplikace využívá.



Obr. 12 Architektura aplikace Logeto
Zdroj: Vlastní zpracování

- **Webová aplikace Logeto** – Webová aplikace Logeto s monolitickou architekturou, která je založená na platformě .NET s pomocí frameworku ASP.NET Web Forms. Aplikace je nasazena ve webovém serveru Internet Information Services.
- **Webová aplikace Accounts** – Webová aplikace Accounts je založena na stejné technologii jako samotná webová aplikace Logeto. Aplikace generuje vyúčtování, provádí platby a sleduje platební transakce na bankovním účtu. Zároveň aplikace poskytuje webové a API rozhraní pro správu účtů a jejich transakcí a sestavuje různé statistiky.
- **Servisní služba –** Služba, která provádí dlouhotrvající úlohy, jako jsou aktualizační skripty databází, hromadný přepočet záznamů, zálohování databází a udržuje transakční kontext pro synchronizaci klientů. Je založena na technologii Windows Communication Foundation (WCF).
- **Redis –** Je open source distribuovaná databáze typu klíč-hodnota. Hodnoty ukládané do databáze je možné jednoznačně identifikovat na základě klíče, který je reprezentován řetězcem. Jedná se o velmi flexibilní systém, který lze škálovat a podporuje různé úrovně perzistence dat na disku. [30] Redis je v tomto případě využíván ve formě vyrovnávací paměti (cache) pro dočasné uložení vybraných dat do operační paměti. K uloženým datům lze následně přistupovat mnohem rychleji než v případě relačních databází.
- **Microsoft SQL Server (MSSQL)** – Relační databázový a analytický systém od společnosti Microsoft [31]. Databázový systém využívá webová aplikace Logeto Accounts a servisní služba pro perzistentní uložení dat. Data uživatelů nejsou uložená v jedné velké databázi, ale pro každý účet je vytvořena separátní databáze. Aktuálně je v databázovém systému cca 9000 databází. Tento přístup zajišťuje vyšší bezpečnost a jednodušší operace s databázemi. Zabrání také uživatelům v případě chyby vidět data z jiných účtů. Databáze lze jednoduše smazat, pokud o to uživatel zažádá, nebo je možné je obnovit na jiném databázovém serveru. Naopak nevýhodou tohoto přístupu je pomalé restartování serveru, obtížné zálohování a obnova databází po havárii, protože nelze využít funkce pro zálohování, které nabízí MSSQL Server např. Always On Availability groups nebo Mirroring. Tyto funkce je

nutné nahradit vlastní implementací. Mezi další nevýhody patří pomalé načítaní databází v SQL Server Management Studio (SSMS). MSSQL Server může být od verze 2017 nainstalovaný na linuxovém serveru, nebo může běžet v kontejneru.

7.3 Vývoj, testování a distribuce

Na vývoji aplikace Logeto se v současné době podílí 13 zaměstnanců. Vývoj je rozdělen mezi dvě vzdálené pobočky, jedna se nachází v Brně a druhá leží v Hradci Králové, kde je také sídlo společnosti. Na pobočce v Brně se nachází pouze oddělení vývojářů, kteří pracují na vývoji webové aplikace a aplikací pro terminál a osobní počítače. V Hradci Králové je zastoupený vývoj mobilních aplikací, zákaznická podpora, a mimo jiné se zde provádí analýza a testování jednotlivých aplikací.

Nová verze produktu je vydávána pravidelně každý měsíc a obsahuje řadu nových funkcí. Zároveň jsou během měsíce vydávané také menší aktualizace, které obsahují pouze drobné opravy problémů. Z tohoto důvodu se musí během měsíce stihnout celý proces vývoje softwaru (plánování, analýza, implementace, testování a nasazení). Pro podporu vývoje softwaru jsou využívány následující nástroje:

- **Jira –** Nástroj pro evidenci úkolů a chyb při vývoji softwaru. Usnadňuje a podporuje proces řízení projektů a požadavků pomocí standardních i agilních metodik. Obsahuje nástroje a funkce pro podporu každé fáze vývojového cyklu. [32]
- **Crucible a Fisheye –** Dva samostatné nástroje, které jsou vzájemně integrované. Fisheye poskytuje možnost procházet, vyhledávat a zobrazit veškeré aktivity ve zdrojovém kódu pro systémy SVN, Git a další, zatímco Crucible nabízí funkce ke kontrole a komentování změn ve zdrojovém kódu (code review). [33][34]
- **Xray –** Kompletní nástroj pro správu testů a podporu řízení manuálního a automatického testování. Je navržen tak, aby každý člen softwarového týmu mohl plánovat, sledovat a vyhodnocovat průběh testování. [35]
- **Teamcity –** Nástroj pro kontinuální integraci od společnosti JetBrains, který podporuje sestavení a nasazení různých typů projektů. [36]

- **Subversion (SVN) –** Centralizovaný systém pro správu a verzování zdrojového kódu. V centralizovaném systému jsou všechny soubory a historická data uložena na jednom centrálním serveru, do kterého vývojáři ukládají své změny. [37]

Vývoj je řízený dle agilní metodiky a je rozdělen do několika iterací. V každé iteraci mají jednotliví vývojáři přiřazený určitý počet úkolů s definovanou prioritou, které mají být v dané iteraci vyřešeny. Úkolem v tomto případě může být oprava nalezené chyby v předchozí iteraci, nebo požadavek implementovat zcela novou funkcionalitu. Po vyřešení všech úkolů je daná iterace uzavřena a je spuštěno sestavení jednotlivých aplikací, které je možné následně nasadit do testovacího prostředí. Během sestavení aplikací jsou automaticky spouštěny jednotkové testy, které spravují samotní vývojáři. V případě selhání těchto testů je nasazení aplikací do testovacího prostředí pozastaveno.

Testování aplikací začíná ihned po nasazení aplikací do testovacího prostředí. Nejprve jsou spouštěny automatické testy, které testují webovou aplikaci, aplikaci pro mobilní telefony s operačním systémem Android a API. Tyto testy mají ověřit základní funkcionalitu a odhalit chyby již na samém počátku testování. Následně jsou mezi testery rozděleny testovací scénáře k manuálnímu otestování. Testovací scénář obsahuje informace o prostředí, ve kterém má být konkrétní scénář proveden, dále podrobný popis s postupy vystihujícími, jak požadovanou funkci a vlastnost aplikace otestovat. Mimo to zahrnuje odkaz na popis úkolu z vývoje pro případný náhled s možností nahlédnout do změn ve zdrojovém kódu. Každý takto vytvořený scénář by měli otestovat všichni testeři, aby se zachytilo co možná nejvíce potencionálních chyb. V případě výskytu chyby je napsán nový úkol, který je přiřazen vývojáři a celá iterace popsaná výše se opakuje do té doby, než jsou veškeré nalezené problémy vyřešeny. Aplikace jsou připraveny k nasazení do produkčního prostředí po kontrole veškerých úkolů, testovacích scénářů a změn ve zdrojovém kódu v repozitáři.

Samotné nasazení webové aplikace do vývojového, testovacího i produkčního prostředí probíhalo před dokončením této závěrečné práce manuálně. To vedlo k dlouhým výpadkům a častým chybám zejména v souvislosti se špatnou

konfigurací nebo rozdílnými prostředími. V produkčním prostředí se aktualizace prováděly pouze v nočních hodinách, aby výpadek zasáhl co nejméně uživatelů. Cílem je tyto problémy odstranit za pomoci kontejnerizace aplikace a automatického nasazení do různých prostředí. Ostatní aplikace jsou nahrávány do distribučních služeb Google Play, Apple Store a Microsoft Store, kde tyto problémy nenastávají.

8 Kontejnerizace webové aplikace Logeto

Následující kapitoly se věnují procesu kontejnerizace již existující webové aplikace Logeto a veškerých komponent, které aplikace využívá. K tomuto účelu byly použity nástroje Docker a Kubernetes. Na základě architektury aplikace půjde především o kontejnerizaci servisní služby a webových aplikací Logeto a Accounts, které jsou založeny na technologiích .Net Frameworku, jakou jsou ASP.NET Web Forms nebo Windows Communication Foundation. Technologie jsou spravovány společností Microsoft a lze je spouštět pouze v kontejnerech Windows, jelikož nejsou multiplatformní. V současné době není možné z časových a finančních důvodů přepsat aplikace do jiné modernější technologie (např. .Net Core), která by umožňovala kontejnerizaci aplikací do Linuxových kontejnerů. Kontejnerizace těchto aplikací je prvním krokem k modernizaci jejich architektury a migraci do cloudu.

Linuxové kontejnery jsou použity především pro Redis cache a další služby pro logování a monitorování celé infrastruktury jako jsou Elasticsearch a Prometheus. Databázový systém MSSQL Server 2017 je nainstalován na samostatném fyzickém serveru. Aktuálně není ze strany Microsoftu doporučeno spouštět databázový systém jako samostatný kontejner v produkčním prostředí.

Výsledkem je jednoduše přenositelná webová aplikace běžící v několika spolu komunikujících kontejnerech, které jsou automaticky nasazeny do vývojového, testovacího a produkčního prostředí. Jejich nasazení a aktualizaci zajistí nástroj pro orchestraci kontejnerů Kubernetes, který podporuje připojení pracovních uzlů s operačními systémy Ubuntu a Windows Server 2019.

8.1 Kontejnerizace aplikace

Kontejnerizace již existující aplikace vyžaduje pro každou komponentu sestavení obrazu, který bude uložen ve veřejném registru, a ze kterého následně bude spuštěn kontejner. Ne ve všech případech je vhodné sestavovat zcela vlastní obraz, ale lze použít obraz z veřejného registru, který byl sestaven jiným uživatelem, nebo společností. Již sestavený obraz byl použit například pro Redis cache. Bohužel u některých obrazů chybí dokumentace nebo Dockerfile, podle kterého byl obraz

sestaven. Nejčastějším důvodem pro vytvoření vlastního obrazu je to, že k hostování aplikace použijeme vlastní kód nebo binární soubory. V takovém případě je nejprve nutné vytvořit soubor Dockerfile pro každý obraz.

8.1.1 Příprava souboru Dockerfile

Dockerfile je jednoduchý textový soubor obsahující seznam instrukcí, které jsou spouštěny během sestavení obrazu. Při psaní těchto souborů je vhodné dodržovat následující doporučená pravidla, která optimalizují rychlost sestavení a velikost obrazu, a zároveň zlepšují čitelnost souboru. [1]

- Obrazy by měly mít co nejmenší velikost, jelikož jsou neustále přesouvány mezi registry a hostiteli, importovány nebo exportovány. Snížení velikosti výsledného obrazu lze docílit odstraněním nadbytečných souborů, nebo minimalizací vrstev. Každá instrukce RUN, COPY a ADD v Dockerfilu vytvoří novou vrstvu v obraze. Seskupením těchto instrukcí do jedné lze docílit snížení počtu vrstev.
- Rychlost sestavení obrazu je možné ovlivnit rozdělením instrukcí na několik samostatných. V případě, že byla stejná instrukce již jednou sestavena, je načtena z mezipaměti, což má za následek snížení výsledného času sestavení.
- Instrukce v Dockerfilu nerozlišují velká a malá písmena, ale z důvodu lepší čitelnosti se používají velká písmena.
- Dlouhé a složité operace lze oddělit znakem zpětného lomítka na více řádků.

Soubory Dockerfile napsané pro každý obraz aplikace zohledňují všechna výše doporučená pravidla a jsou verzované stejným způsobem jako vyvíjené aplikace ve verzovacím systému SVN. Na následující ukázce je Dockerfile pro sestavení obrazu webové aplikace Logeto.

```
ARG WINDOWS_VERSION
FROM logeto.azurecr.io/iis:${WINDOWS_VERSION}

ARG VERSION

COPY monitoring.ps1 /

RUN $url=[string]::Format('https://systemartreleases.blob.core.
     windows.net/releases/{0}/ServerWeb-{0}.zip',
      $env:VERSION.Replace('.','_')) ; \
    Invoke-WebRequest -UseBasicParsing -Uri $url -Out
     'ServerWeb.zip' ; \
    Expand-Archive -Path 'ServerWeb.zip' -DestinationPath 'Logeto'
     -Force; \
    Remove-Item -Path 'ServerWeb.zip' -Recurse -Force

RUN Import-Module WebAdministration ; \
    New-WebAppPool -Name 'Logeto' ; \
    New-Website -Name 'Logeto' -Port 80 -ApplicationPool 'Logeto'
     -PhysicalPath 'C:\Logeto' ; \
    icacls 'Windows\Temp' /grant 'IIS AppPool\Logeto:(OI)(CI)F' /T

ENTRYPOINT Start-Process powershell.exe -ArgumentList '-file
            C:\monitoring.ps1' ; \
           C:\ServiceMonitor.exe w3svc
```

Zdrojový kód 6 Dockerfile webové aplikace Logeto
Zdroj: Vlastní zpracování

První skupina řádků v Dockerfilu deklaruje, jaký základní obraz bude použit pro sestavení obrazu. V tomto případě se jedná o základní obraz *logeto.azurecr.io/iis* s tagem předaným jako argument. V tomto obrazu jsou povoleny některé funkce systému Windows (například IIS, URL Rewrite nebo vzdálený přístup) a nainstalovány další programy třetích stran, jako je podpora Brotli komprese. Další instrukce zkopíruje powershell skript pro monitorování procesu IIS (w3wp.exe). V případě, že proces využívá příliš operační paměti, nebo webová stránka neodpovídá, je proveden úplný dump paměti procesu pro další diagnostiku a proces je restartován. Důležitými řádky v tomto souboru jsou řádky s instrukcí RUN, které spouští jednotlivé příkazy v powershellu a rozdělují je do logických celků. Příkazy stáhnou ze soukromého úložiště konkrétní verzi aplikace a nasadí ji do webového

serveru IIS. Poslední instrukce spustí sledování procesu IIS uvnitř kontejneru až v momentě, kdy se kontejner spustí z vytvořeného obrazu.

8.1.2 Sestavení kontejnerů

V momentě, kdy jsou pro každou aplikaci napsané soubory Dockerfile, lze na jejich základě sestavit příslušné obrazy. Předpokladem pro sestavení těchto obrazů je správně nakonfigurované prostředí, které musí současně podporovat sestavení obrazů pro Windows i Linux kontejnery. Tyto kontejnery nejsou vzájemně kompatibilní, jelikož kontejnery sdílejí jádro hostitelského operačního systému. Není tedy možné sestavit obrazy pro Linuxové kontejnery ve Windows, stejně tak není možné sestavit obrazy pro Windows kontejnery na Linuxu.

Sestavení obrazů pro Linux a Windows kontejnery vyžaduje dva nakonfigurované systémy s operačními systémy Linux a Windows, ve kterých je nainstalován nástroj Docker. Další možností, která je dostupná pouze na systémech s operačním systémem Windows je experimentální funkce LCOW (Linux containers on Windows). Funkce podporuje, díky virtualizaci, sestavení obrazů pro Linux kontejnery na Windows a je vhodná zejména pro vývojová prostředí. Sestavení obrazů nástrojem Docker je možné pomocí příkazu *docker build*, kterému stačí předat správnou cestu k souboru Dockerfile.

Na základě problému se složitou konfigurací a údržbou lokálního prostředí byla pro sestavení obrazů vybrána cloudová služba Azure Containers Registry. Služba podporuje automatické sestavení obrazů pro Linux i Windows kontejnery, globální replikaci a je možné ji integrovat do různých vývojářských nástrojů. Minimálním požadavkem je nainstalování rozhraní příkazového řádku Azure CLI, které je dostupné pro Windows, Linux i macOS. [38]

Následující příkaz sestaví obraz *web:8.0.3.2043_1809* pro webovou aplikaci Logeto verze 8.0.3.2043 na základě výše vytvořeného Dockerfilu uloženého v adresáři Web.

```
az acr build \
      --registry logeto \
      --image web:8.0.3.2043_1809 \
      --platform windows \
      --build-arg WINDOWS_VERSION=1809 \
      --build-arg VERSION=8.0.3.2043_1809 \
       .\web
```

Zdrojový kód 7 Sestavení obrazu pro webovou aplikaci Logeto
Zdroj: Vlastní zpracování

Sestavené obrazy jsou automaticky uloženy v soukromých repozitářích ve veřejném registru Logeto pro pozdější sdílení a využití. Těmito repozitáři jsou *logeto.azurecr.io/web*, *logeto.azurecr.io/service* nebo *logeto.azurecr.io/accounts*. V registru lze obrazy jednoduše spravovat přes webový portál Microsoft Azure. Aktuálně je vytvořeno 12 soukromých repozitářů, ve kterých je uloženo několik stovek obrazů různých verzí. Názvy těchto obrazů v soukromých repozitářích se později definují v konfiguračních souborech Kubernetes. Kubernetes tyto obrazy stáhne a vytvoří z nich kontejnery na příslušných uzlech clusteru kdykoliv to je potřeba.

8.2 Nasazení aplikace do clusteru Kubernetes

Nástroj pro orchestraci kontejnerů Kubernetes zajišťuje nasazení kontejnerizovaných aplikací a jejich nepřetržitý běh. V případě havárie je schopen kontejnery automaticky obnovit na jiné části infrastruktury a udržet tak aplikace co nejdostupnější. Kubernetes se neustále vyvíjí a každá aktualizace přináší nové funkce. Od verze 1.14 je možné připojit do clusteru pracovní uzel s operačním systémem Windows Server 2019. To umožnilo využít tento nástroj pro orchestraci kontejnerů v produkčním i testovacím prostředí, které vyžaduje hybridní cluster pro nasazení Windows a Linux kontejnerů.

8.2.1 Příprava prostředí

Kubernetes verze 1.16 je provozován přímo na lokálních virtuálních strojích, jejichž specifikace jsou uvedeny v následující tabulce. Pro zajištění vysoké dostupnosti jsou tyto virtuální stroje rovnoměrně rozmístěny mezi dva fyzické servery.

	UBUNTU 18.04.4 LTS (BIONIC BEAVER)		**WINDOWS SERVER 2019**	
HOSTNAME	k8s-vm	k8s-01-vm	k8s-win-vm	k8s-01-win-vm
ROLE	Master	Worker	Worker	Worker
CPU	8 jader	8 jader	8 jader	8 jader
RAM	24 GB	24 GB	8 GB	8 GB
SSD	500 GB	500 GB	250 GB	250 GB

Tabulka 1 Specifikace virtuálních strojů
Zdroj: Vlastní zpracování

Kubernetes cluster tvoří jeden řídící (master) a tři pracovní (worker) uzly. Řídící uzel může být nakonfigurován pouze na uzlech s operačním systém Linux. Není tedy možné mít cluster složený pouze z uzlů s operačním systém Windows. Na každém z těchto uzlů jsou stažené binární soubory kubectl, kubelet a kube-proxy a nainstalovaný nástroj Docker, který poskytuje standard pro vytváření a distribuci kontejnerů. Samotná konfigurace lokálního hybridního clusteru vyžaduje provést několik kroků:

- **Konfigurace řídícího uzlu pomocí nástroje Kubeadm** – Kubeadm automatizuje instalaci a konfiguraci komponent Kubernetes, jako je API server nebo Controller Manager. Nevytváří však uživatele ani neinstaluje další závislosti na úrovni operačního systému. V hybridním prostředí je důležité brát také v úvahu, že některé prostředky Kubernetes jsou ve výchozím nastavení rozmístěny na jakékoliv uzle. V takovém případě není žádoucí, aby prostředky pro Linux byly umístěny na uzlech s operačním systémem Windows a naopak. Prostředky je možné rozmístit na konkrétní

uzle pomocí štítků a selektorů (např. *kubernetes.io/os=linux* nebo *kubernetes.io/hostname=k8s-vm)*.

- **Instalace síťového pluginu Flannel** – Platforma jako je Kubernetes přepokládá, že každý pod (skupina jednoho nebo více kontejnerů) má svou vlastní IP adresu. Kontejnery na jednom uzlu mohou snadno komunikovat prostřednictvím lokálního rozhraní. Komunikace mezi pody je složitější a vyžaduje samostatnou síťovou komponentu, která může směřovat provoz z podu na jednom uzlu do podu na druhém. Tuto funkci poskytuje síťový plugin Flannel. [39]
- **Připojení pracovních uzlů Windows a Linux –** Připojení pracovních uzlů do clusteru vyžaduje provedení jednoho příkazu na každém uzlu. Tento příkaz obsahuje nezbytné informace o clusteru, jako jsou IP adresa a port řídícího uzlu nebo bezpečností klíč.

Úspěšné ověření konfigurace Kubernetes je možné pomocí nástroje kubectl příkazem *kubectl get nodes*. Pokud všechny uzle mají ve sloupci Status hodnotu Ready, je cluster připraven ke spuštění libovolné kontejnerizované aplikace.

```
NAME            STATUS   ROLES    AGE   VERSION
k8s-01-vm       Ready    <none>   35d   v1.16.0
k8s-01-win-vm   Ready    <none>   14d   v1.16.2
k8s-vm          Ready    master   51d   v1.16.0
k8s-win-vm      Ready    <none>   51d   v1.16.2
```

Obr. 13 Seznam uzlů v clusteru Kubernetes
Zdroj: Vlastní zpracování

Z důvodu složitější instalace a údržby clusteru na lokální infrastruktuře bude v budoucnu produkční prostředí přesunuto do cloudové služby Azure Kubernetes Service (AKS), která umožňuje rychle vytvořit a spravovat Kubernetes cluster. Zároveň tato služba také podporuje nasazení Windows i Linux kontejnerů.

8.2.2 Nasazení aplikace

Nasazení všech komponent webové aplikace do prostředí Kubernetes vyžaduje napsání konfiguračních souborů pro každou aplikaci, které definují požadovaný stav. V Kubernetes je možné všechny prostředky definovat pomocí konfiguračních souborů (tzv. manifestů) ve formátu YAML. YAML je textový formát, který je čitelný nejen strojem, ale i člověkem a přináší řadu výhod [26].

- **Komfort** – Není nutné přidávat všechny parametry do příkazového řádku.
- **Údržba** – Soubory YAML je možné zařadit do systému pro správu zdrojového kódu a mít tak možnost sledovat všechny provedené změny.
- **Flexibilita** – Pomocí YAML souborů je možné vytvářet mnohem složitější struktury, než by bylo možné v příkazovém řádku.

Na následujících ukázkách jsou konfigurační soubory YAML pro webovou aplikaci Logeto. Jedná se o konfigurační soubory pro prostředky Deployment, Service a ConfigMap. Deployment představuje požadovaný stav skutečného nasazení v Kubernetes. Obvykle obsahuje všechny požadované informace, jako je obraz kontejnerizované aplikace, počet požadovaných replik nebo umístění konfiguračních souborů v ConfigMaps a Secrets. V tomto případě se očekává nasazení pouze jedné repliky webové aplikace Logeto z obrazu *logeto.azurecr.io/web:8.0.3.2043_1809*. Tento obraz je uložen v soukromém repozitáři ve veřejném registru Logeto, z toho důvodu musí definice obsahovat přístupové údaje k tomuto registru, aby bylo možné obraz stáhnout. Přihlašovací údaje jsou uloženy v šifrované podobě v prostředku Secret, který umožňuje ukládat citlivé informace v Kubernetes. Deployment umístí pod na uzel na základě štítků a selektorů s otevřeným portem 80. Soubor je zakončen definicí adresáře, ve kterém po spuštění kontejneru budou vytvořeny soubory definované v ConfigMap.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  namespace: logeto-app
  labels:
    app: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      name: web
      labels:
        app: web
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: kubernetes.io/os
                  operator: In
                  values:
                  - windows
                - key: logeto.com/host
                  operator: In
                  values:
                  - secondary
      imagePullSecrets:
      - name: azure-container-registry
      containers:
      - name: web
        image: logeto.azurecr.io/web:8.0.3.2043_1809
        ports:
        - name: http
          containerPort: 80
        volumeMounts:
        - name: config
          mountPath: "C:/Logeto/Configuration"
      volumes:
      - name: config
        configMap:
          name: web
```

Zdrojový kód 8 Kubernetes manifest – Deployment
Zdroj: Vlastní zpracování

Definicí Service je možné zpřístupnit webovou aplikaci Logeto jiným aplikacím uvnitř clusteru. V tomto případě služba přesměrovává provoz z portu 80 na cílový port kontejnerů, který odpovídá štítkům v selektoru.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: logeto-app
  labels:
    app: web
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
  selector:
    app: web
```

Zdrojový kód 9 Kubernetes manifest – Service
Zdroj: Vlastní zpracování

ConfigMap umožňuje oddělit data, která jsou závislá na prostředí, od kontejnerizovaných aplikací. To jim umožňuje zůstat jednoduše přenositelnými napříč různými prostředími. Webová aplikace Logeto vyžaduje několik konfiguračních souborů, ve kterých jsou uloženy informace potřebné pro připojení k MSSQL Serveru, Redisu, servisní službě nebo webové aplikaci Accounts. Níže uvedený ConfigMap se používá k ukládání informací týkajících se MSSQL Serveru.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: web
  namespace: logeto-app
  labels:
    app: web
data:
  Connections.config: |-
    <connectionStrings>
      <add name="LogetoConnectionString"
           connectionString="Data Source=192.168.1.5;
                             Initial Catalog=master;
                             Persist Security Info=True;
                             User ID=sa;
                             Password=Gate2Db;
                             MultipleActiveResultSets=True;
                             Max Pool Size=512"
           providerName="System.Data.SqlClient" />
      <add name="DefaultConnection"
           connectionString="Data Source=192.168.1.5;
                             Initial Catalog=ASPState;
                             Persist Security Info=True;
                             User ID=sa;
                             Password=Gate2Db;
                             MultipleActiveResultSets=True"
           providerName="System.Data.SqlClient" />
    </connectionStrings>
```

Zdrojový kód 10 Kubernetes manifest – ConfigMap
Zdroj: Vlastní zpracování

Vytvoření těchto prostředků probíhá provedením příkazu *kubectl apply -f <cesta ke konfiguračnímu souboru YAML>*. Výsledkem je nasazená webová aplikace Logeto v clusteru Kubernetes. Tento postup je nutné zopakovat pro všechny komponenty, které aplikace využívá. Informace o nasazených kontejnerizovaných aplikacích pro produkční prostředí a jejich stavu lze získat příkazem *kubectl get pods --namespace logeto-app.*

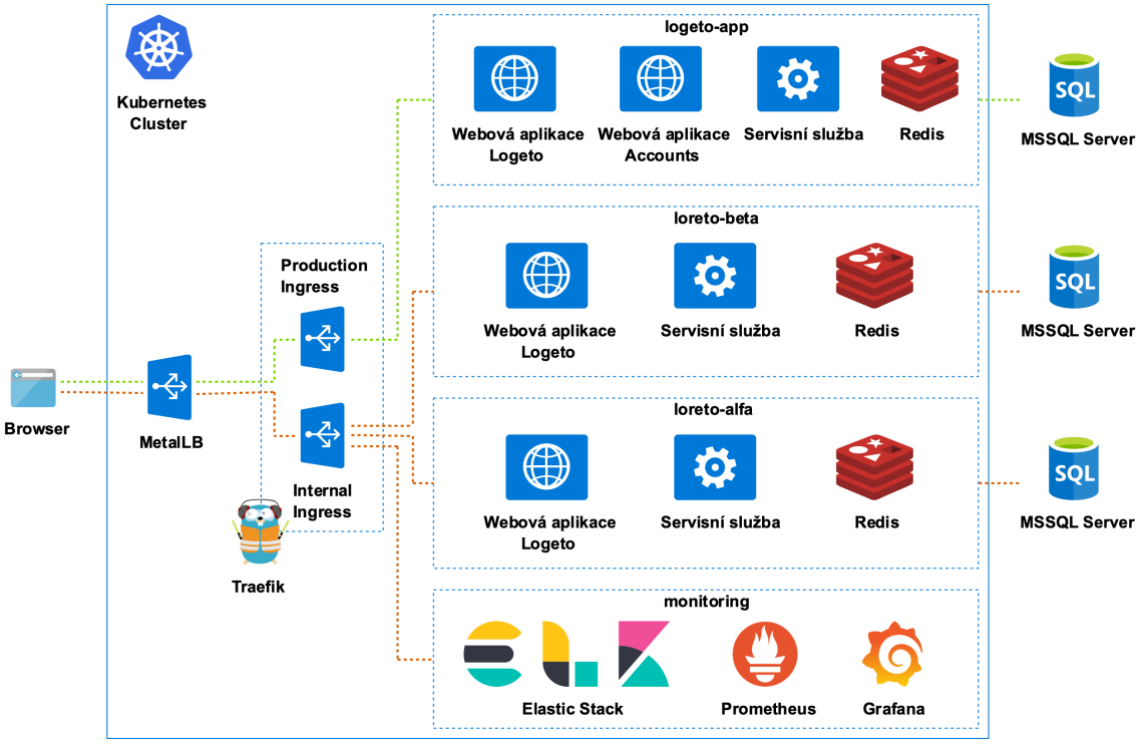
```
NAME                          READY   STATUS    RESTARTS   AGE
redis-0                       1/1     Running   0          10h
service-76587cc97f-8c5ql      1/1     Running   0          10h
web-845597b7fd-sdnpp          1/1     Running   0          10h
accounts-78cd9f684f-8z6vg     1/1     Running   0          10h
```

Obr. 14 Seznam podů v Kubernetes
Zdroj: Vlastní zpracování

Pro zpřístupnění aplikací z vnějšku clusteru je využíván Ingress Controller Traefik a MetalLB. Traefik je nasazen ve dvou instancích, z nichž jedna zpracovává pouze příchozí požadavky do produkčního prostředí. MetalLB umožňuje v clusteru, který není provozován v cloudovém prostředí, vytvářet služby typu LoadBalancer [40].

V rámci clusteru je vytvořeno několik virtuálních clusterů (namespaces). Využívají se pro oddělení produkčního, testovacího a vývojového prostředí, ve kterých jsou nasazené různé verze stejné webové aplikace, včetně veškerých komponent, které aplikace využívá (Redis, Servisní služba a Accounts). Zároveň jsou využity pro oddělení nástrojů pro monitoring a logování.

Následující obrázek poskytuje úplný pohled na celou infrastrukturu, která zobrazuje kontejnerizované aplikace nasazené do clusteru Kubernetes, včetně jeho okolních systémů.



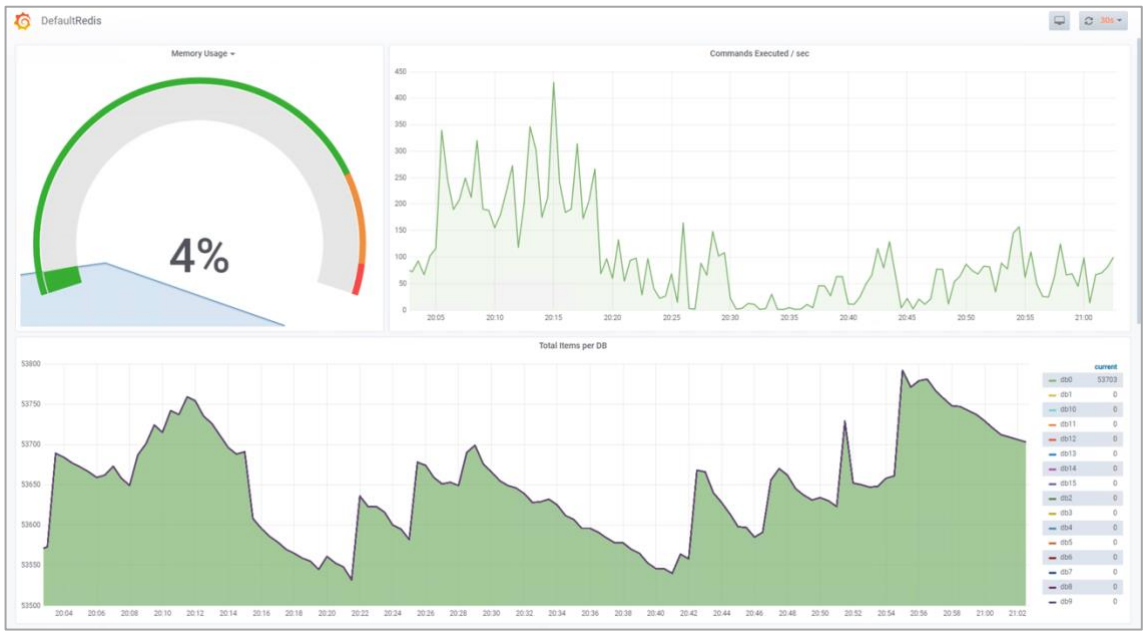
Obr. 15 Kubernetes cluster
Zdroj: Vlastní zpracování

8.2.3 Monitoring a logování

Monitorování a logování kontejnerizovaných aplikací v Kubernetes vyžaduje zcela jiný přístup, než jaký je v případě tradičních aplikací běžících přímo na fyzických a virtuálních strojích. V těchto prostředích je přesně známo, na jakém stroji je daná aplikace nasazená. To neplatí v prostředí Kubernetes. V tomto prostředí kontejnerizované aplikace mohou být kdykoliv spuštěny, odstraněny, škálovány nebo přesunuty na jiný uzel. Kromě toho se může kdykoliv změnit počet těchto uzlů.

Existuje několik nástrojů, které sledují a vyhodnocují metriky a logy kontejnerizovaných aplikací spuštěných v clusteru Kubernetes. Nejzákladnějším nástrojem, který je nejjednodušší na instalaci, je Kubernetes Dashboard. Zobrazuje pouze základní metriky související se statistikami využití paměti a CPU na jednotlivých uzlech a kontejnerech. V tomto případě se jedná o pokročilejší nástroje Prometheus a Elastic Stack (ELK).

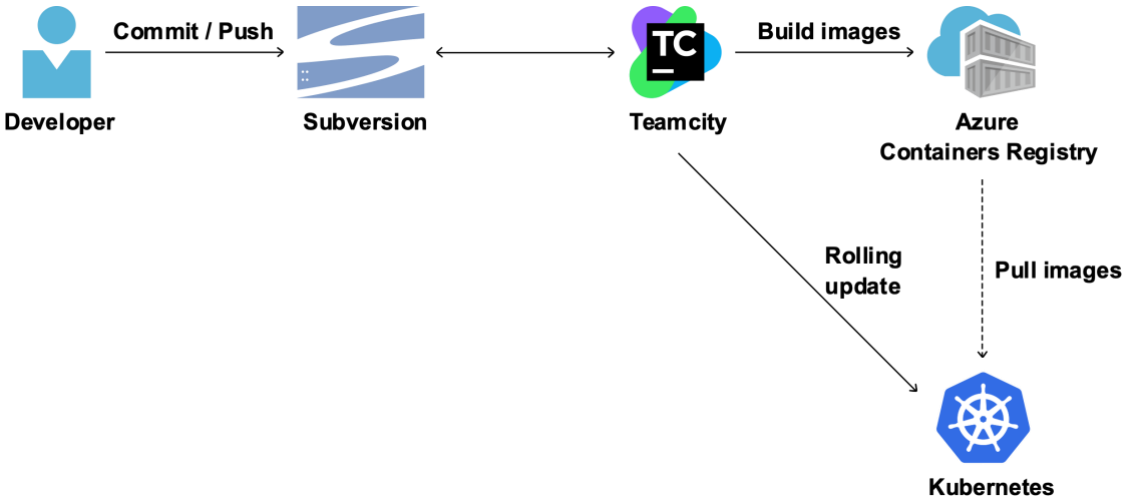
- **Elastic Stack** – Sada nástrojů (Elasticsearch, Logstash a Kibana), která efektivně zpracovává, ukládá a zpřístupňuje logy z různých aplikací. Elasticsearch je škálovatelné úložiště, do kterého je možné uložit jakýkoliv dokument ve formátu JSON. Logstash shromažďuje logy z různých zdrojů, které před odesláním do Elasticsearch zpracuje. Na základě konfiguračního souboru logy filtruje, nebo k nim přiřadí další informace, například zjistí stát podle IP adresy, ze které se klient připojuje k webové aplikaci. Posledním nástrojem je Kibana, která načítá data z Elasticsearch a z nich vytváří různé vizualizace (grafy, tabulky, dashboardy, atd.). [41]
- **Prometheus –** Moderní nástroj pro monitorování, který umožňuje sbírat metriky z různých běžících aplikací nebo virtuálních a fyzických strojů. Metriky je možné následně filtrovat a analyzovat, přičemž výsledky jsou vizualizovány nástrojem Grafana. V případě, že dojde k určité události (například, když se zvýší využití CPU), používá Prometheus další nástroj Alertmanager, který zašle upozornění na Slack, webhook nebo e-mail. [42]



Obr. 16 Grafana dashboard
Zdroj: Vlastní zpracování

8.3 Automatizace sestavení a nasazení

Současně s kontejnerizací webové aplikace Logeto bylo nutné zavést určitou formu automatizace nasazení do různých prostředí. Každý člen v týmu má možnost prostřednictvím nástroje Teamcity sestavit obrazy jakékoliv verze aplikace a nasadit je do Kubernetes. Tento proces se skládá z několika kroků, které jsou definované pomocí powershell skriptů.



Obr. 17 Automatizace sestavení a nasazení
Zdroj: Vlastní zpracování

1. Vývojáři provedou změny ve zdrojovém kódu a uloží je do repozitáře na serveru Subversion.
2. V Teamcity se zahájí sestavení jednotlivých aplikací a spustí se jednotkové testy.
3. Pokud předchozí krok proběhne v pořádku, jsou z příslušných souborů Dockerfile sestaveny obrazy v cloudové službě Azure Containers Registry. Obrazy jsou následně uloženy v soukromých repozitářích ve veřejném registru Logeto.
4. Posledním krokem je aktualizace webové aplikace ve vývojovém, testovacím, nebo produkčním prostředím. Příkaz *kubectl rolling update* stáhne aktuální obraz z registru a aktualizuje kontejnerizovanou aplikaci v Kubernetes. Během tohoto kroku se dále provádí promazání testovacích databází nebo promazání CDN cache.

5. Webová aplikace je dostupná přes webový prohlížeč na adrese https://app.logeto.com (produkční prostředí).

Výsledkem je automatizovaný proces, který umožňuje neustále aktualizovat webovou aplikaci Logeto. Celý proces od nahrání zdrojového kódu vývojářem po aktualizaci kontejnerů v Kubernetes trvá v průměru 40 minut. Samotná aktualizace kontejnerů ale trvá pouhých 40 sekund.

9 Závěr

Tato práce naplnila všechny své předpokládané cíle, a sice seznámila přehledně čtenáře s problematikou kontejnerizačních technologií. Kromě popisu těchto technologií poskytuje práce také ucelený přehled doplněný o řadu názorných obrázků a zdrojových kódů.

V teoretické části byly detailně popsány základní technologie, jako je například Docker, který se stal standardem pro vývojáře a společnosti pracující s kontejnerizovanými aplikacemi. Docker nabízí různé nástroje pro správu životního cyklu kontejnerů a jejich komponent, jak je blíže popsáno v příslušné kapitole. Mimo jiné práce představila další nástroje pro orchestraci kontejnerů, jako například Docker Swarm nebo Kubernetes, které jsou důležité pro automatizaci a správu velkého počtu kontejnerů v různých prostředích.

Na základě teoretických předpokladů byla úspěšně provedena kontejnerizace již existující webové aplikace Logeto a veškerých jejích komponent, čímž byl naplněn hlavní cíl práce. Aplikace a její komponenty jsou vyvíjeny společností Systemart s.r.o. v programovacím jazyce C# za použití technologií, jako jsou ASP.NET WebForms, WCF, IIS atd. Proces kontejnerizace obnášel sestavení obrazů pro každou komponentu dle doporučených pravidel. Tyto obrazy byly následně uloženy ve veřejném registru Azure Containers Registry. Vzhledem k tomu, že byly použity Windows i Linux kontejnery, bylo potřeba správně nakonfigurovat různá prostředí ve kterých byla aplikace nasazena. Pro nasazení kontejnerizované aplikace byl nakonfigurován cluster Kubernetes, který právě podporuje oba typy kontejnerů. Tento cluster je monitorován nástroji Prometheus a Elasticsearch. Celý proces sestavení a nasazení kontejnerizované aplikace byl zautomatizován, což umožnilo jednoduché a neustálé nasazování nových verzí do různých prostředí. Díky tomu se doba nasazení nové verze snížila na pouhých několik sekund a výpadek aplikace se téměř nedotkne koncových uživatelů.

Do budoucna je plánováno přesunutí produkčního prostředí do cloudu. K tomu bude využita cloudová služba Azure Kubernetes Service (AKS). Důvodem k tomuto přesunu je odstranění komplikací s instalací a údržbou clusteru na lokální infrastruktuře.

10 Seznam použité literatury

[1] *Windows and containers* [online]. Microsoft, 2019 [cit. 2020-01-18]. Dostupné z: https://docs.microsoft.com/en-us/virtualization/windowscontainers

[2] MELL, Peter a Timothy GRANCE. *The NIST Definition of Cloud Computing* [online]. 2011 [cit. 2020-01-18]. Dostupné z: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[3] ROUNTREE, Derrick a Ileana CASTRILLO. *The basics of cloud computing: understanding the fundamentals of cloud computing in theory and practice*. Boston: Elsevier/Syngress, [2014]. ISBN 978-0-12-405932-0.

[4] LASZEWSKI, Tom a Prakash NAUDURI. *Migrating to the cloud: Oracle client/server modernization*. Boston: Elsevier/Syngress, c2012. ISBN 978-1-59749-647-6.

[5] Co je cloud computing?: Škálovatelný. Flexibilní. Inteligentní. Objevte sílu cloudu. *Oracle Česká republika: Integrated Cloud Applications and Platform Services* [online]. Oracle, 2019 [cit. 2020-01-18]. Dostupné z: https://www.oracle.com/cz/cloud/what-is-cloud-computing

[6] Co je cloud computing?: Průvodce pro začátečníky. *Microsoft Azure* [online]. Seattle: Microsoft, 2020 [cit. 2020-01-18]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-cloud-computing

[7] REESE, George. *Cloud application architectures: [building applications and infrastructure in the Cloud]*. Sebastopol, CA: O'Reilly Media, c2009. ISBN 978-0-596-15636-7.

[8] PISCAER, Joep. *Kubernetes in the Enterprise* [online]. Bluffton: ActualTech Media, 2019 [cit. 2020-01-18]. Dostupné z: https://platform9.com/resource/the-gorilla-guide-to-kubernetes-in-the-enterprise

[9] TURNBULL, James. *The Docker Book* [online]. 2017 [cit. 2020-01-26]. Dostupné z: https://dockerbook.com/

[10] KANE, Sean P. a Karl MATTHIAS. *Docker: up and running*. Second edition. Sebastopol, CA: O'Reilly Media, [2018]. ISBN 978-1-492-03673-9.

[11] *AbcLinuxu.cz - Linux na stříbrném podnose* [online]. [cit. 2020-01-26]. Dostupné z: http://www.abclinuxu.cz/

[12] STONEMAN, Elton. *Docker on Windows: From 101 to production with Docker on Windows.* Packt Publishing, 2017. ISBN 9781785281655.

[13] MCCABE, John a Michael FRIIS. *Introduction to Windows Containers* [online]. Redmond: Microsoft Press, 2017 [cit. 2020-01-26]. Dostupné z: http://download.microsoft.com/download/a/1/3/a13a2b9e-d47c-4f93-b180-f7f9cd3382a7/introduction_to_containers_ebook.pdf

[14] NICKOLOFF, Jeff. *Docker in action* [online]. Shelter Island, NY: Manning Publications, Co., [2016] [cit. 2020-01-26]. ISBN 978-163-3430-235.

[15] GOASGUEN, Sébastien. *Docker cookbook*. Sebastopol, CA: O'Reilly Media, 2015. ISBN 978-1-491-91971-2.

[16] *Azure Service Fabric documentation* [online]. 2020 [cit. 2020-01-26]. Dostupné z: https://docs.microsoft.com/en-us/azure/service-fabric/

[17] *Container Technology Wiki* [online]. Aqua Security, 2018 [cit. 2020-01-26]. Dostupné z: https://www.aquasec.com/wiki

[18] *Developer Survey Results 2019* [online]. 2019 [cit. 2020-01-26]. Dostupné z: https://insights.stackoverflow.com/survey/2019

[19] *Docker Documentation: Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries.* [online]. Docker, 2020 [cit. 2020-01-26]. Dostupné z: https://docs.docker.com/

[20] GABRIEL, Dr. a N. SCHENKER. *Containerize your Apps with Docker and Kubernetes: Deploy, scale, orchestrate, and manage containers with Docker and Kubernetes*. Birmingham: Packt, 2018. ISBN 978-1-78961-036-9.

[21] *Docker: Empowering App Development for Developers* [online]. Docker, 2020 [cit. 2020-01-26]. Dostupné z: https://www.docker.com/

[22] *CURRENTS: A quarterly report on developer trends in the cloud* [online]. DigitalOcean, 2018 [cit. 2020-01-26]. Dostupné z: https://www.digitalocean.com/assets/media/currents-research/pdf/DigitalOcean-Currents-Q2-2018.pdf

[23] *Container Management: Kubernetes vs Docker Swarm, Mesos + Marathon, Amazon ECS* [online]. Platform9, 2018 [cit. 2020-01-26]. Dostupné z: https://platform9.com/wp-content/uploads/2018/08/kubernetes-comparison-ebook.pdf

[24] *Kubernetes Documentation* [online]. The Linux Foundation, 2020 [cit. 2020-01-26]. Dostupné z: https://kubernetes.io/docs/home/

[25] *Google Cloud: Google Kubernetes Engine documentation [online]. [cit. 2020-01-26]. Dostupné z: https://cloud.google.com/kubernetes-engine/docs*

[26] *Mirantis: Blog* [online]. Mirantis [cit. 2020-01-26]. Dostupné z: https://www.mirantis.com/blog/

[27] *Logeto - Attendance and Time tracking* [online]. Czech Republic: Systemart, 2018 [cit. 2020-03-23]. Dostupné z: https://www.logeto.com

[28] BURDA, Tomáš. *Testování softwaru* [online]. Hradec Králové, 2017 [cit. 2020-03-23]. Dostupné z: https://theses.cz/id/dsbtq2/. Bakalářská práce. Univerzita Hradec Králové, Fakulta informatiky a managementu, Hradec Králové. Vedoucí práce Doc. RNDr. Petra Poulová, Ph.D.

[29] *Dokumentace Výkazu práce* [online]. Hradec Králové: Systemart, 2020 [cit. 2020-03-23]. Dostupné z: https://dokumentace.vykazprace.cz

[30] *Redis* [online]. Redis Labs [cit. 2020-03-23]. Dostupné z: https://redis.io/

[31] *SQL Server technical documentation* [online]. Microsoft, 2020 [cit. 2020-03-23]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

[32] *Jira: Software pro sledování požadavků a projektů* [online]. Atlassian, 2020 [cit. 2020-03-23]. Dostupné z: https://www.atlassian.com/cs/software/jira

[33] *Fisheye: Vyhledávání kódu a nástroj diff pro systémy SVN, Git a další* [online]. Atlassian, 2020 [cit. 2020-03-23]. Dostupné z: https://www.atlassian.com/cs/software/fisheye

[34] *Crucible: Nástroj k revizi kódu pro Git, SVN, Perforce a další* [online]. Atlassian, 2020 [cit. 2020-03-23]. Dostupné z: https://www.atlassian.com/cs/software/crucible

[35] *Xray: Cutting Edge Test Management for Jira* [online]. Xpand IT, 2020 [cit. 2020-03-23]. Dostupné z: https://www.getxray.app

[36] *TeamCity: the Hassle-Free CI and CD Server by JetBrains* [online]. JetBrains, 2020 [cit. 2020-03-23]. Dostupné z: https://www.jetbrains.com/teamcity/

[37] *Apache Subversion* [online]. The Apache Software Foundation, 2018 [cit. 2020-03-23]. Dostupné z: https://subversion.apache.org

[38] *Microsoft Docs: Azure Container Registry* [online]. Microsoft, 2020 [cit. 2020-03-23]. Dostupné z: https://docs.microsoft.com/cs-cz/azure/container-registry/

[39] *Coreos/flannel: Flannel is a network fabric for containers, designed for Kubernetes* [online]. 2020 [cit. 2020-03-24]. Dostupné z: https://github.com/coreos/flannel/

[40] *MetalLB* [online]. 2020 [cit. 2020-03-24]. Dostupné z: https://metallb.universe.tf

[41] *Elastic: Open Source Search: The Creators of Elasticsearch, ELK Stack & Kibana* [online]. United States: Elasticsearch B.V., 2020 [cit. 2020-03-24]. Dostupné z: https://www.elastic.co

[42] *Prometheus* [online]. The Linux Foundation, 2020 [cit. 2020-03-24]. Dostupné z: https://prometheus.io

11 Přílohy

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení:	**Bc. Tomáš Burda**
Osobní číslo:	**I1700481**
Adresa:	Na Vinici 313, Vysoké Veselí, 50703 Vysoké Veselí, Česká republika
Téma práce:	Využití Dockeru pro správu aplikací v kontejnerech
Téma práce anglicky:	Using Docker to manage applications in containers
Vedoucí práce:	doc. RNDr. Petra Poulová, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cílem práce je převod aplikací běžících ve firemním prostředí do kontejnerů, včetně automatického nasazení do produkčního prostředí.

1. Úvod
2. Teoretická část
   - 2.1. Virtualizace
   - 2.2. Docker
     - i. Architektura
     - ii. Obrazy
     - iii. Kontejnery
   - 2.3. Orchestrace kontejnerů
     - i. Docker compose
     - ii. Docker Swarm
     - iii. Kubernetes
   - 2.4. Nasazení kontejnerů v cloudu
     - i. Azure
     - ii. AWS
   - 2.5. Monitoring kontejnerů
3. Praktická část
   - 3.1. Analýza aplikací
   - 3.2. Migrace aplikací do kontejnerů
4. Závěr

Seznam doporučené literatury:

STONEMAN, Elton. *Docker on Windows: From 101 to Production with Docker on Windows*. Birmingham: Packt, 2017. ISBN 9781785281655.

MCCABE, John a Michael FRITS. *Introduction to Windows Containers* [online]. Washington: Microsoft Press, 2017 [cit. 2018-10-14]. Dostupné z: https://blogs.msdn.microsoft.com/microsoft_press/2017/08/30/free-ebook-introduction-to-windows-containers

Podpis studenta: Burda Datum: 23. 4. 2020

Podpis vedoucího práce: Datum: 23. 4. 2020