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

# Analýza a návrh multi-cloud platformy pro geograficky distribuovanou webovou aplikaci

Diplomová práce

**Autor:** Bc. Radek Vašátko

**Studijní obor:** Informační management

**Vedoucí práce:** Ing. Jakub Pavlík, MSc.

Hradec Králové duben, 2018

## Prohlášení

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

V Hradci Králové dne 25. dubna 2018 Radek Vašátko

## Poděkování

Rád bych zde poděkoval Ing. Jakubu Pavlíkovi, MSc. za odborné vedení práce, podnětné rady a čas, který mi věnoval.

## Anotace

Cílem této diplomové práce je návrh multi-cloud platformy a následná implementace geograficky distribuované webové aplikace založené na technologii Docker kontejnerů s využitím automatického průběžného dodání. Práce vysvětluje základní principy a modely cloud computingu, rozlišení mezi Standard/Legacy a nativními cloud aplikacemi, transformaci monolitických aplikací do mikroslužeb a podstatu multi-cloud prostředí a Docker kontejnerů. V další části jsou popsány kontejnerové orchestrační nástroje a řešení pro průběžné dodávání softwaru do prostředí multi-cloudu. Praktická část je zaměřena na popis zvolené aplikace pro implementaci, a především na přípravu multi-cloud prostředí u dvou vybraných veřejných poskytovatelů cloudu – Google Cloud Platform a Microsoft Azure s vytvořením Kubernetes clusterů. Nakonec je provedeno nasazení webové aplikace na Kubernetes clustery v prostředí multi-cloudu a otestováno automatické průběžné dodání.

## Annotation

**Analysis and design of multi-cloud platform for geographically distributed web applications**

The goal of the thesis is a proposal of a multi-cloud platform and also the implementation of geographically distributed web application which is based on the technology of the Docker containers with using continuous delivery. The thesis explains basic principles and models of cloud computing, differences between Standard/Legacy and Cloud Native application. Moreover, it focuses on the transformation of monolithic applications to microservices and last but not least the essence of multi-cloud environment in the cases of two public providers of cloud, specifically Google Cloud Platform and Microsoft Azure with the creation of Kubernetes clusters and the installation of a multi-cloud continuous delivery platform Spinnaker. Finally, there is realized the implementation of web application on Kubernetes clusters in multi-cloud and testing of automatic continuous delivery.

# Obsah

# Seznam obrázků

# 1 Úvod

V současné době téměř každý uživatel internetu, aniž by si to uvědomoval, využívá služeb cloud computingu, například při spolupráci na on-line dokumentech, synchronizaci dat nebo prohlížení webu, který v cloudu běží. V důsledku rostoucích rychlostí internetového připojení je možné přistupovat a pracovat s velkým množstvím dat uložených ve vzdáleném pronajatém prostředí. Požadované prostředky jsou tak přiděleny pouze v případě potřeby, což vede k vysoké efektivnosti a nízkým nákladům.

Nicméně není příliš známo o implementaci aplikací do prostředí cloudu natožpak multi-cloudu, kdy jde o využití více poskytovatelů. Vzhledem k tomu, že existuje mnoho rozdílů mezi lokálním a cloud prostředím, nemohou být při procesu nasazení aplikací použity stejné techniky. Kvůli složitosti dnešních webových aplikací se na jejich vývoji zpravidla podílí tým několika vývojářů a zároveň je obtížné přenášet tyto aplikace a její součásti do jiného prostředí nebo na jiný počítač, než ve kterém byly vyvíjeny. To je způsobeno velkým počtem používaných operačních systémů, programovacích jazyků, nástrojů nebo odlišných nastavení.

Z toho vyplývá, že když je třeba spustit vlastní kód v jiném prostředí, než ve kterém byl vyvíjen, je nutné docílit identického nastavení jako v prostředí předchozím. To není vždy snadné a už vůbec ne rychlé. Proto byla vyvinuta technologie kontejnerů, která velice usnadňuje testování a nasazení v různých prostředích.

Dalším problémem, který mnohé softwarové společnosti řeší je dostupnost aplikace. Díky výhodám multi-cloud prostředí a implementace aplikace na více poskytovatelů cloudu je možné tomuto problému předejít například tak, že aplikace bude geograficky rozmístěna po celém světě a u různých, na sobě nezávislých poskytovatelů. Přínosem je také, že aplikace je dostupná na serveru co nejblíže ke koncovému uživateli a v případě výpadku jednoho ze serverů je provoz přesměrován na jiný.

Diplomová práce v teoretické části vysvětluje principy a modely cloud computingu, rozdíly mezi monolitickými aplikacemi a mikroslužbami, podstatu multi-cloud prostředí

a kontejnerového nástroje Docker. V další kapitole jsou popsány nástroje pro orchestraci mikroslužeb a řešení pro průběžné dodávání softwaru do multi-cloudu s podrobnou charakteristikou vybraných nástrojů Kubernetes a Spinnaker. Hlavním cílem této diplomové práce je návrh multi-cloud platformy a nasazení vybrané webové aplikace založené na technologii Docker kontejnerů s využitím automatického průběžného dodání a multi-cloud prostředí pro zajištění její geografické dostupnosti. Praktická část je zaměřena na vymezení zvolené aplikace pro implementaci, přípravu multi-cloud prostředí v Google Cloud Platform a Microsoft Azure s vytvořením dvou Kubernetes clusterů a dále na instalaci všech potřebných nástrojů. Nakonec je otestováno automatické průběžné dodávání webové aplikace na Kubernetes clustery v prostředí multi-cloudu.

# 2 Struktura cloudových aplikací

Cloud Computing je v posledních letech velmi populární a je považován za další klíčový milník ve vývoji informačních technologií. Přestože pojem cloud je možné slýchat v současné době ze všech možných směrů, tak pro mnoho lidí je abstraktním pojmem a prakticky nevědí, co si pod tímto slovem představit.

## 2.1 Cloud computing

Cloud computing, často nazývaný pouze jako „Cloud“, je model, který závisí především na sdílení zdrojů a služeb přes internet namísto vyřizování žádostí od lokálních serverů či jednotlivých zařízení. Cloud může poskytovat pestrou škálu funkcí přes internet, jako například úložiště, virtuální počítače a servery, softwarové aplikace a další. Firmy již nenásledují tradiční model, ve kterém nakoupí vlastní servery a postaví serverové místnosti, ale počítačový výkon si objednávají jako službu od specializovaných dodavatelů. Cloud computing tak představuje nový způsob myšlení ohledně počítačových zdrojů. [1]

Cloud computing se člení podle dvou základních modelů:

- Distribuční modely cloud computingu
- Modely nasazení cloud computingu

### 2.1.1 Distribuční modely cloud computingu

Distribuční model definuje, co je v rámci služby nabízeno, zpravidla je to software nebo hardware a také jejich kombinace.

### Infrastruktura jako služba - IaaS

Infrastruktura jako služba je model, kdy poskytovatel nabízí uživatelům pronájem virtuální IT infrastruktury, jako jsou např. servery, datová úložiště, síťové prvky, firewally a další v podobě služby za pravidelné poplatky. Vybraná služba je dostupná okamžitě a je možné snadno měnit její technické parametry. U IaaS se tak uživatelé nemusí starat o správu fyzických serverů, monitoring, zabezpečení a další, veškerá správa je na poskytovateli služby. [1]

### Platforma jako služba - PaaS

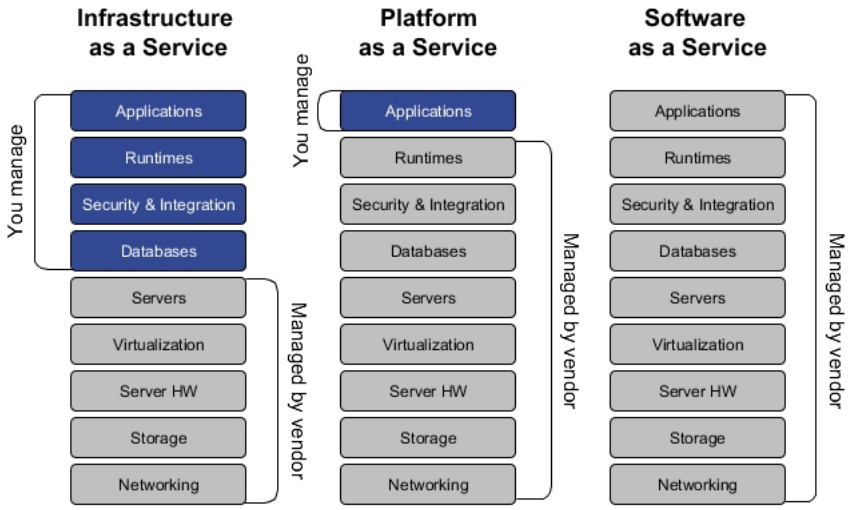
V rámci tohoto modelu nabízí poskytovatel platformu, která slouží pro provoz aplikací vyvíjených zákazníkem. Platformou se v tomto případě rozumí kompletní aplikační nebo vývojové prostředí, ve kterém je možné vyvíjet vlastní aplikace.

Poskytovatel služby PaaS tedy poskytuje všechny výpočetní zdroje hardwarové i softwarové povahy, které technologicky umožňují fungování platformy a zákazník využívá tuto platformu pouze pro vývoj a test svých aplikací. PaaS je navržen tak, aby podporoval celý životní cyklus webové aplikace - sestavení, testování, nasazení, správu a aktualizace. [2]

Cloud platformy umožňují do několika minut nasadit síťové aplikace nebo služby a zpřístupnit je komukoliv na světě. V případě, že se aplikace stane více využívanou, cloud ji snadno upraví tak, aby zvládla větší provoz. Cloud platformy představují další krok ve vývoji informačních technologií, který se zaměřuje výlučně na aplikace a data bez ohledu na infrastrukturu. [1]

### Software jako služba - SaaS

V distribučním modelu software jako služba je vybraná aplikace (software) poskytována přes internet uživateli jako služba na vyžádání. Jednu instanci této služby, která běží v cloudovém prostředí, může využívat i více uživatelů pomocí tzv. tenkých klientů. Uživateli je tedy zprostředkován pouze přístup k aplikacím, ale ne aplikace samotné. Odpadá nutnost koupě, instalace, aktualizace a správy softwaru i hardwaru. [1]




**Obrázek 2.1:** Distribuční modely cloud computingu (převzato z [2])

### 2.1.2 Modely nasazení cloud computingu

Modely nasazení cloud computingu představují přesné kategorie prostředí, ve kterých lze cloud provozovat. Tyto modely reprezentují způsob a rozsah sdílení výpočetní infrastruktury, které využívají služby technologie cloud computingu z pohledu uživatele (zákazníka).

#### Privátní cloud

Privátní cloud, také známý jako interní cloud, je prostředí pro cloud computing, kde jsou výpočetní služby nabízené přes internet nebo privátní interní síť jenom vybraným uživatelům, ne veřejnosti. Privátní cloud poskytuje vyšší úroveň zabezpečení a ochrany osobních údajů prostřednictvím bran firewall a interního hostování, čímž zajišťují, že citlivá data nejsou přístupná externím uživatelům. Nevýhodou jsou vyšší náklady na provoz a údržbu. [3]

#### Veřejný cloud

Je druh cloud hostingu, ve kterém jsou požadované služby poskytovány ve virtualizovaném prostředí a přístupné prostřednictvím veřejného internetu a to i více uživatelům. Nabízí rychlejší nasazení než privátní cloud a téměř neomezenou škálovatelnost. Za správu a údržbu celého systému zodpovídá poskytovatel cloudových služeb. [3]

#### Komunitní cloud

Označuje prostředí, ve kterém jsou služby vzájemně sdílené mezí více organizacemi, které patří do určité skupiny - komunity. Tyto organizace může spojovat například bezpečnostní politika nebo podobný obor zájmu. Komunitní cloud vzniká sdružením prostředků vlastněných touto skupinou členů a následně jsou prostředky poskytovány zpět formou služby stejné komunitě. [3]

#### Hybridní cloud

Je cloud, který kombinuje veřejný cloud a privátní cloud a umožňuje mezi nimi sdílení dat a aplikací. Je tedy kombinací výše zmíněných modelů nasazení. Využívá různé typy vzájemně propojených služeb od různých poskytovatelů a zároveň možnost privátního cloudu. Hybridní cloud nabízí vyšší úroveň flexibility než samostatné modely veřejného nebo privátního cloudu. Organizace díky tomu získají potřebný výpočetní výkon veřejného cloudu pro základní a méně citlivé výpočetní úlohy, zatímco důležité aplikace a data zůstanou v místním prostředí, bezpečně ukryté za firewallem. [3]

## 2.2 Standard/Legacy aplikace vs. nativní cloud aplikace

Legacy aplikace je jakákoli aplikace založená na starších technologiích a hardwaru, jako jsou např. sálové počítače, které nadále poskytují organizacím základní služby. Tyto aplikace jsou často velké, monolitické a obtížně spravovatelné. Proto se v současné době více využívají nativní cloud aplikace a zároveň probíhá proces transformace z Standard/Legacy aplikací na cloud aplikace za účelem snížení provozních nákladů a především efektivním zajištění aktuálních a budoucích požadavků. [4]

### 2.2.1 Standard/Legacy aplikace

Neexistuje žádná jednoznačná definice, která by určovala, jaké aplikace jsou typu Standart/Legacy. Ale obvykle zahrnují následující atributy:

- Jsou uložené na lokálním počítači nebo serveru, přičemž využívají lokální úložiště.

- Na jednom zařízení (serveru) mají spuštěno mnoho služeb.
- Instalace a aktualizace potřebují velké množství skriptů a manuálních procesů, které jsou často špatně dokumentované.
- Konfigurace je uložena v souborech na několika místech a promíchána s aplikačními soubory.
- Meziprocesová komunikace využívá lokální systém.
- Jsou navržené k tomu, že pouze jedna instance dané aplikace bude spuštěna na jednom serveru. [5]

Z čehož vyplývá, že tyto aplikace mají mnoho nevýhod jako například - velmi obtížné automatické nasazení, v případě výpadku serveru delší doba opětovného spuštění aplikace, nasazení nových verzí je zdlouhavý manuální proces, který není snadné vrátit zpět, často dochází k rozdílnosti testovacího a produkčního prostředí, nelze je jednoduše škálovat a další. [5]

Standart/Legacy aplikace jsou již poměrně zastaralé a mohou být nestabilní kvůli problémům s kompatibilitou se stávajícími operačními systémy nebo prohlížeči. Ale stále je tento typ aplikací velmi využíván, protože většinou plní kritické funkce a není jednoduché je převést na nový a efektivní typ. [4]

### 2.2.2 Nativní cloud aplikace

Jsou aplikace určené přímo pro chod v cloudu, tudíž jsou odděleny od fyzické infrastruktury. Skládají se z mnoha služeb, přičemž každá služba je variabilní, odolná a kombinovaná. Tyto aplikace jsou vytvořené pro rychlé nasazení v prostředí cloudu a nabízejí uživatelům lepší dostupnost a bezpečnost. Používají pružnou infrastrukturu, jsou dostupné přes API rozhraní (Application Programming Interface), musejí být schopné škálovat rychlým tempem a jsou autonomní.

Jednou z největších výhod nativních cloud aplikací je schopnost nabízet téměř neomezenou výpočetní sílu na vyžádání spolu s moderními datovými a aplikačními službami. Dalším znakem, který se rychle stává standardem těchto aplikací, je využívaní mikroslužeb a Docker kontejnerů. [6]

### 2.2.3 Cattle vs. Pets

Je určitá analogie (koncept) porovnání stylů využívání IT prostředků s tím, jak se chováme ke zvířatům v domácnosti (Pets) ve srovnání s dobytkem (Cattle). Jde o to, že domácí zvířata jsou pro nás jedinečná, vychovávaná a považovaná za člena rodiny, tudíž potřebují více času a péče. Kdežto dobytek je chápán jako stádo, které vyžaduje méně opatrování a je brán jako komodita, tedy něco, s čím lze obchodovat. [7]

#### Cattle

Jsou především virtuální servery, které jsou vytvořeny za použití automatizovaného nástroje a jestliže selže jeden nebo i více z nich, jsou nahraditelné. Typicky při výpadku není nutný žádný lidský zásah do systému. Mezi nejčastější využití patří webové aplikace. [8]

#### Pets

Představují servery, které jsou nepostradatelné nebo jedinečné a nemohou být tak nikdy poškozeny či vypnuty. Jestliže se nějakým způsobem poškodí, je nutné je opravit. Je tedy nezbytné se o tato zařízení neustále starat. [8]

#### Porovnání klíčových vlastností

**Cattle:**

- Automatické škálování - při zvýšení zátěže se automaticky přidělí odpovídající výkon.
- Navrženo pro případ selhání (architektura Design-for-failure) - v případě výpadku jednoho nebo i více ze serverů dojde k samostatnému nahrazení identickým.
- Sdílená architektura - distribuovaná architektura, v níž je každé zařízení v síti nezávislé a soběstačné. V celém systému žádné zařízení nesdílí totožné úložiště nebo paměť. Podstatnou výhodou je odstranění jakéhokoliv bodu selhání v dané síti, čímž je zajištěn bezproblémový chod všech služeb.

**Pets:**

- Podpora NIC Bonding - umožňuje spojit více síťových rozhraní do jednoho kanálu a tím zvýšit dostupnou šířku pásma. Po propojení se dvě nebo i více fyzických zařízení jeví jako jedno a mají společnou MAC adresu.

- Nutná pravidelná správa serverů.
- 100% pokrytí všech požadavků.
- Nezbytná vysoká dostupnost dat - poskytuje snížení výpadků aplikací a ztráty dat.
- Trvalé blokové ukládání dat - permanentně uchovává uložená data. [9]

## 2.3 Monolitické aplikace a mikroslužby

Při vývoji aplikací existují dva různé typy přístupů, a to vytváření tzv. monolitických aplikací nebo tvorba prostřednictvím mikroslužeb. Monolitické aplikace představují souhrnné řešení, které je vytvářeno s jasně definovaným záměrem, zatímco mikroslužby rozdělují architekturu dané aplikace na jednotlivé služby, které jsou vyvíjeny samostatně. [6]

### Monolitické aplikace

Monolitická aplikace je nejčastěji tvořena souvislým kódem v jediném souboru a software je navržen tak, aby fungoval samostatně. Jednotlivé komponenty aplikace jsou vzájemně závisle spojeny, a aby mohl být kód správně zpracován nebo kompilován, musejí být všechny komponenty součástí dané aplikace.

V případě, že je potřeba využívat jen určitou část aplikace, je jedinou možností pouze vyjmout část požadovaného kódu a ten poté přizpůsobit, aby byl funkční samostatně. Pozdější úpravy kódu jsou tedy úměrné délce a složitosti kódu. Tento způsob je tak vhodný především pro jednoduché aplikace. [10]

Výhody:

- Prosté testování - zpravidla je zde menší počet prvků a proměnných, proto je testování snadnější.
- Lepší propustnost aplikace.

Nevýhody:

- Každá sebemenší chyba může způsobit pád celé aplikace.
- V případě vydání nové verze nebo oprav chyb je nutné přepsat celou aplikaci a poté znovu testovat a nasadit.

- U starších aplikací často chybí někdo, kdo by znal všechny souvislosti aplikace, znají pouze určitou její část, o kterou se starají.
- Při škálování je možné duplikovat pouze celou aplikaci s veškerými, často i nepotřebnými funkcemi. [10] [11]

### Mikroslužby

Architektura mikroslužeb strukturuje aplikaci jako kolekci volně spojených služeb a umožňuje nepřetržité dodávání a nasazování velkých, komplexních aplikací.

Mikroslužby jsou založeny na rozdělení aplikace na malé samostatné služby, kde každá služba zpravidla zajišťuje jednu konkrétní oblast dané aplikace. Jednotlivé služby mezi sebou komunikují prostřednictvím protokolů (nejčastěji HTTP). Hlavním cílem architektury je tedy rozdělit monolitické aplikace na více menších služeb, přičemž každá uvnitř obsahuje vlastní architekturu a poté lze tuto dekompozici zužitkovat. [6]

Architektura mikroslužeb přináší mnoho výhod:

- Škálování - mikroslužby poskytují mnohem rychlejší a snadnější škálovatelnost než monolitické aplikace, a to z důvodu, že není nutné škálovat celou aplikaci. Je možné určit pouze vybrané služby, které škálovat potřebují a současně je klonování zvolených služeb zcela nezávislé na ostatních.
- Rychlé a nezávislé nasazení - každou službu lze spravovat a upravovat nezávisle, čímž je zajištěno velmi rychlé nasazení.
- Při použití architektury mikroslužeb je velká monolitická aplikace rozložena na množinu menších spravovatelných služeb, které jsou lépe udržitelné.
- Znovupoužitelnost - jednotlivé mikroslužby mohou být v budoucnu využívány i jinými klienty, zařízeními.
- Robustnost - pokud se objeví chyba v některé z mikroslužeb, neznamená to automaticky pád celé aplikace. Ostatní služby se o chybu postarají a zajistí chod aplikace. Nicméně na tyto chyby je potřeba obstarat reakci služeb.
- Organizace - při vývoji aplikace založené na mikroslužbách se může každý tým vývojářů soustředit pouze na jednu či více mikroslužeb. Vznikají tak menší vývojové týmy, ve kterých je lepší kooperace a vývoj je tak rychlejší.

Použitím mikroslužeb vzniká distribuovaná aplikace, proto je potřeba konfigurace

a správa dalších prvků, které u monolitických není nutné řešit. V jistém smyslu to lze považovat za určité nevýhody této architektury:

- Komunikace - poněvadž jsou služby izolovány od jiných, je třeba mezi nimi zajistit komunikaci. V monolotické aplikaci je komunikace prováděna na úrovni programového volání metod, oproti tomu u mikroslužeb se komunikuje nejčastěji pomocí API rozhraní. Komunikace mezi službami je logicky pomalejší než u monolitické architektury, kde aplikace působí jako celek. Při komunikaci se musí uskutečnit zakódování zprávy, její odeslání, přijetí, dekódování a následné zpracování. Často je také nutné zaslat totožným způsobem i odpověď.
- Testování - testování jednotlivých mikroslužeb je poměrně jednoduché a je to jednou z výhod této architektury. Avšak testování celé aplikace je náročnější, protože je potřeba mít spuštěné všechny ostatní služby, které testovaná služba používá. Což může být v případě složité aplikace obtížné.
- Pořadí při nasazení - jak již bylo zmíněno, každou službu lze nasazovat nezávisle na ostatních, ovšem někdy může nastat situace, že změna provedená v jedné službě se promítne do ostatních, proto je třeba při nasazení myslet na správné pořadí mikroslužeb. [6] [11]

## 2.4 Multi-cloud řešení

Multi-cloud je prostředí založené na používání více kombinací veřejných, privátních nebo i hybridních cloudů, ale zároveň nemusí nutně zahrnovat různé typy cloudů. Společnost se může rozhodnou využívat více privátních nebo veřejných cloudů od různých poskytovatelů, jako např. Google Cloud Platform, Amazon Web Services, OpenStack, Microsoft Azure a další.[12] Každý z těchto poskytovatelů nabízí celou řadu služeb z oblasti cloud computingu, přičemž některá služba může běžet lépe a cenově výhodněji na jedné platformě, zatímco ostatní služby na jiné. Společnosti využívající multi-cloud si tak mohou vybrat optimální technologie a služby od nejrůznějších poskytovatelů cloudu, aby vytvořily nejlepší dostupné řešení. [13]

Jde tedy o využití více cloudových služeb v heterogenním prostředí primárně za účelem snížení závislosti na jednom poskytovateli, zvýšení flexibility nebo zmírnění dopadů a rizik při výpadcích.

Výhody multi-cloudu:

- Minimalizace rizik: multi-cloud přináší velmi užitečné zmírnění rizik, k čemuž přispívá jednoduché a rychlé zotavení po výpadku, možnost redundance dat a co nejmenší počet jednotlivých bodů selhání. Diverzifikace služeb mezi více poskytovatelů tak pomáhá proti výpadkům aplikace nebo ztrátě dat.
- Vysoká dostupnost a výkon: je zajištěna díky rozložení pracovní zátěže mezi více poskytovatelů.
- Geografická dostupnost: řešení poskytovatelů cloudových služeb jsou obsluhována z geograficky odlišných lokalit, což znamená, že je vybírán přípojný bod co nejblíže k uživateli kdekoli na celém světě.
- Snížení nákladů: více smluv s několika poskytovateli zvyšuje složitost, ale také může výrazně snížit náklady. V multi-cloudu platí obecné pravidlo cloud computingu, že se platí pouze za to, co se opravdu využívá a díky více poskytovatelům přidává schopnost hledat a vyjednat nejlepší možnou cenu. [14] [15]

### 2.4.1 Poskytovatelé cloud platformy

#### Google Cloud Platform - GCP

Služba Google Cloud Platform, kterou nabízí společnost Google, je zaměřena především na firemní uživatele. Poskytuje velké množství modulárních cloudových služeb jako například virtuální počítače, Kubernetes Engine, úložiště dat, práci s Big Daty, strojové učení, komplexní prostředí pro vývoj aplikací a mnoho dalších. [16]

#### Amazon Web Services - AWS

Cloudové služby od Amazonu se skládají z několika částí, které může uživatel využívat nezávisle na sobě. V současné době poskytuje tato společnost vysoce spolehlivou, škálovatelnou a nízkonákladovou platformu cloudové infrastruktury. Služby zahrnují vývojářské nástroje, datová úložiště, kontejnery, síťovou infrastrukturu, databázové nástroje, aplikační služby a další. [17]

### Microsoft Azure

Cloudová platforma od společnosti Microsoft je kolekcí integrovaných cloudových služeb, mezi které patří služby pro vývoj, správu a analýzu aplikací, databázové nástroje, datové úložiště a další. Podporuje velké množství operačních systémů, programovacích jazyků a databází. Uživatel může využít řadu nástrojů pro tvorbu nových aplikací nebo využít již vytvořené aplikace. [18]

### OpenStack

OpenStack je open-source platforma, která umožňuje provozovat cloud typu IaaS i na běžně dostupném hardwaru. OpenStack se skládá ze série vzájemně nezávislých projektů, které byly spojeny dohromady, a tím poskytuje různé komponenty, ze kterých je složeno celé cloudové řešení. Jednou z hlavních motivací OpenStack projektu bylo umožnit organizacím vytvořit a nabídnout cloudové služby, které je možné provozovat na standardním hardwaru. Klíčovou vlastností je, že výpočetní, síťové i ukládací služby zajišťuje software, nikoli hardware. [19]

## 2.5 Docker

Je open-source platforma nabízející odlehčený druh virtualizace pomocí linuxových kontejnerů. Docker vychází z tradičních linuxových distribucí, jako je Ubuntu nebo Red Hat, a umožňuje zabalit aplikace a služby do bitových kopií, které běží ve svých vlastních přenositelných kontejnerech. Tyto kontejnery je poté možné poměrně jednoduše přesouvat mezi fyzickými nebo virtuálními prostředími bez potřeby nějaké úpravy. Docker je tedy nástroj, který usnadňuje práci s virtuálními kontejnery.

Docker tímto způsobem nabízí velmi vysoký stupeň přenositelnosti aplikací a vyhovuje tak vysoce škálovatelným aplikacím. Povaha Dockeru dále umožňuje spouštění jedné služby nebo aplikace v každém jednotlivém kontejneru namísto určité sady. Docker proto významně zkracuje čas a odstraňuje problémy při šíření aplikace skrz jednotlivé vývojové týmy nebo při přesunu aplikace z testovacího do produkčního prostředí. [20]

Kontejner je vnímán jako tzv. obálka pro proces, který byl spuštěn. Přirovnání s klasickými přepravními kontejnery je zde vhodné, poněvadž přepravce se nemusí starat o vnitřek kontejneru a vidí jenom standardní rozhraní, se kterým umí pracovat. Stejné výhody

mají kontejnery virtuální, a tak díky konzistentnímu API není důležité, jaká technologie právě běží uvnitř daného kontejneru. Hlavním základním prvkem jsou sestavené obrazy (images) různých linuxových distribucí, nad kterými lze spouštět jednotlivé procesy. Proces běží v uzavřeném prostředí - kontejneru, ale je spuštěn jako jakýkoliv jiný, tudíž kontejner nelze chápat jako samostatně běžící systém, nýbrž jako proces běžící z libovolného uzavřeného prostředí. [21]

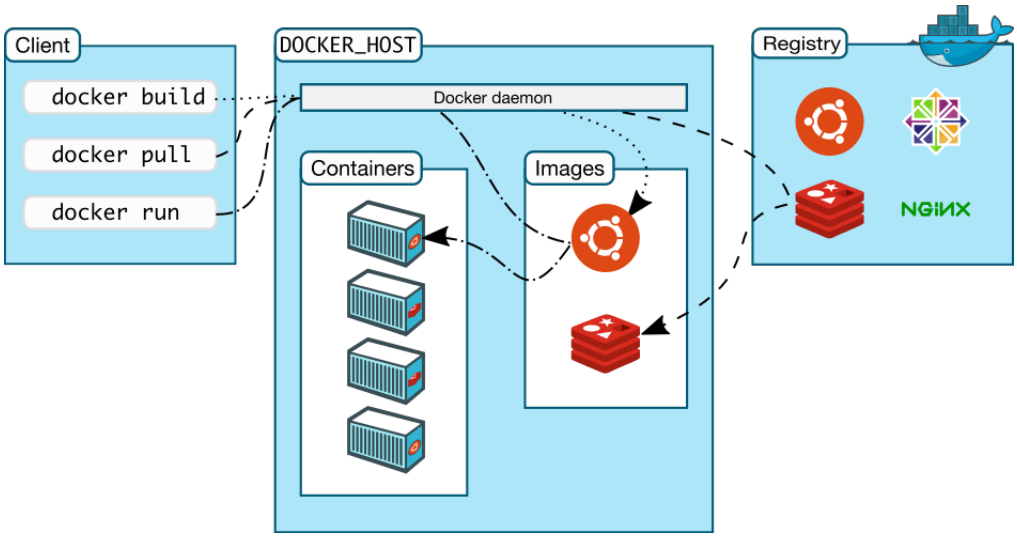
Díky nástroji Docker je prostředí odolné a jednodušší na napodobení, má tedy takové vlastnosti, že vícenásobné provedení má vždy stejný výsledek. Tím odpadá častý problém, že na jednom zařízení je daná věc funkční a na jiném nikoli. Není tedy nutné řešit konfiguraci aplikace v novém prostředí, protože vše potřebné ke spuštění je zabalené spolu s aplikací v kontejneru.

Další výhodou je, že kontejnery startují v podstatě okamžitě (v řádu milisekund) a kontejnerový Docker Engine umožňuje mít spuštěno až desítky kontejnerů současně. Je ale nutné dodržovat základní pravidlo Dockeru - do jednoho kontejneru patří jen jedna věc. To znamená, nikdy nedávat do jednoho kontejneru více aplikací, nevznikne tak problém s tím, že by dvě aplikace měly konfliktní závislosti.

Jak již bylo zmíněno výše, Docker je open-source, z čehož vyplývá, že v podstatě kdokoliv se může účastnit na jeho vývoji a přidávat tak další funkce, které požaduje, ale zatím nejsou k dispozici. Docker je dostupný pro operační systémy Linux, Windows a macOS. Kontejnery jsou vzájemně nezávislé na hardwaru i operačním systému, který běží na hostitelském počítači - lze tedy kontejner nakonfigurovaný například na Linux spustit v Dockeru běžícím na Windows a opačně. [20]

### 2.5.1 Docker architektura

Docker využívá architekturu klient-server oddělující klienta a server, kteří spolu komunikují. V tomto případě Docker klient komunikuje s Docker daemonem, který obstarává výpočetně obtížné požadavky jako je např. nasazování, spouštění a distribuce kontejnerů. Docker klient i server mohou běžet na stejném systému (počítači) nebo se lze skrz klienta připojit ke vzdálenému Docker daemonu. Po připojení spolu oba, klient i daemon, komunikují prostřednictvím REST API přes UNIX sokety nebo síťové rozhraní. [21]




**Obrázek 2.2:** Docker architektura [21]

### Docker Engine

Docker Engine je základní součást Dockeru, která umožňuje vytváření, spouštění, běh a správu samotných kontejnerů. Tyto úkoly má konkrétně na starosti Docker daemon, který běží na pozadí hostitelského systému a jeho ovládání je umožněno pomocí Docker klienta. [21]

### Klient

Docker klient je primární uživatelské rozhraní pro platformu Docker. Přijímá příkazy od uživatele a komunikuje s Docker deamonem nejčastěji přes HTTP protokol. Jeden klient může komunikovat s více vzájemně nesouvisejícími deamony. [21]

### Daemon

Docker deamon běží na hostitelském systému a jak již bylo zmíněno výše, uživatel používá docker klienta pro komunikaci s deamonem. Je to stálý proces, který spravuje jednotlivé kontejnery. Je možné spustit více kontejnerů s několika parametry.

Pro deamona i klienta Docker používá různé binární soubory. Docker deamon se řídí příkazy od Docker Remote API. [21]

### Docker image

Docker image je obraz souborového systému a v podstatě se jedná o data, která jsou dostupná pouze pro čtení (práva read-only), není možné ho spustit a slouží pouze jako počáteční stav nového kontejneru. Je to šablona s instrukcemi pro vytvoření daného kontejneru. Takový obraz může například obsahovat operační systém Ubuntu nebo Debian s webovým serverem a nainstalovanou webovou aplikací. [21]

Image se skládá z několika vrstev, proto Docker používá Union File System (UnionFS), který tyto vrstvy spojí do jednoho uceleného obrazu. UnionFS dovoluje složkám a souborům, které jsou odděleny systémovým souborem, aby byly dostatečně transparentně překryty a vytvořily jeden úplný souborový systém.

V případě, že je nutná změna image, například z důvodu aktualizace na novou verzi, vytvoří se nová vrstva, která je ale pouze aktualizací vrstvy stávající. Tudíž není třeba šířit zcela nový image, ale jen novou vrstvu aktualizace. Distribuce image je tak mnohem rychlejší a méně komplikovaná.[20]

### Kontejner

Docker kontejner používá linuxové jádro hostitelského počítače a skládá se z metadat a dalších mnoha souborů, které lze přidat při vytváření a jsou potřebné pro správný běh zvolených aplikací. [20]

Kontejner je tedy spustitelná instance Docker image. Je možné jej spustit, spustit na pozadí, zastavit, přesunout nebo vymazat pomocí Docker API nebo příkazů v příkazovém řádku. V případě, že Docker spustí kontejner z vybraného image, přidá UnionFS novou vrstvu (pro čtení i zápis) na vrchol image, ve které poté daná aplikace běží. [21]

### Registr

Docker registr je jakousi knihovnou Docker image, poněvadž tyto image uchovává. Registr může být soukromý nebo veřejný a také může být na stejném serveru jako deamon nebo klient, nebo na zcela samostatném serveru.

Soukromý registr uchovává soukromé image určité organizace a běží zpravidla za firewallem. Příkladem veřejného registru je Docker Hub, což je oficiální repozitář image pro Docker a obsahuje velké množství již existujících a okamžitě dostupných obrazů, které si

může každý uživatel stáhnout. Zároveň je možné do tohoto veřejného registru přidat svůj vlastní image. Image jsou dostupné zdarma nebo za určitý poplatek. [21]

Image z veřejného repozitáře si může stáhnout kdokoli do lokálního úložiště za pomocí příkazu - *docker pull*. Poté je možné na základě takového obrazu spustit kontejner nebo vytvořit nový image a použít stažený obraz jako základ. [20]

#### Dockerfile

Dockerfile je textový soubor neboli skript, ve kterém jsou zapsané příkazy a instrukce jejichž prostřednictvím se vytvoří nový Docker image. Soubor a tvorbu image lze spustit příkazem - *docker build*. V Dockerfile je tedy definované, jaké parametry bude nový image zahrnovat. O vznik samotného obrazu se stará daemon, který všechny příkazy zpracuje v přesném pořadí, v jakém následují a následně Docker image vytvoří. [20]

### 2.5.2 Další typy kontejnerové virtualizace

Kromě Dockeru, který v kontejnerovém řešení hraje klíčovou roli, existují i další jako například Rocket a LXD.

#### Rocket

Rocket je kontejnerové prostředí obsažené v open-source operačním systému CoreOS. CoreOS je postavený na linuxovém jádře a jedná se o speciálně vyvíjený systém pro poskytování infrastruktury na nasazování na clustery. Rocket je navržený především pro prostředí s velkým počtem serverů, kde se počítá se značným objemem dat, a proto je zaměřen na rychlost a zabezpečení.

Rocket se skládá ze dvou částí, kdy každá z nich je vytvořena jako samostatný nástroj s příkazovým řádkem. Jedná se o Actool a Rkt. Actool zabezpečuje vytváření kontejnerů a jejich následnou kontrolu. Rkt načítá a spouští image. Je pojmenovaný zkrácením názvu Rocket, protože všechny hlavní linuxové příkazy používané nástrojem Rocket se skládají z těchto tří písmen. [22]

### LXD

LXD je nástroj pro správu kontejnerů a zároveň daemon poskytující API rozhraní, který je velmi podobný jako v Docker a Rocket, a umožňuje pracovat s LXC kontejnery.

Nástroj LXD je schopný spouštět plně funkční operační systémy v kontejnerech stejně jako fyzické nebo virtuální stroje. Tyto typy kontejnerů jsou většinou určené pro chod na dlouhý časový úsek a nejčastěji jsou vytvářené na nové systémové image, které ještě nebyly upravované. [23]

Oproti tomu Docker je více zaměřen na nasazování malých kontejnerů, které se v případě aktualizace snadno vymění za nové. Docker se tímto způsobem více podobá klasickému konceptu distribuce softwaru než ostatní nástroje na správu kontejnerů.

# 3 Nástroje pro Multi-cloud delivery

Nasazení aplikací do cloudových prostředí zahrnuje průběžnou integraci, průběžné dodání a průběžné nasazení softwaru. Průběžná integrace často závisí na psaní a testování softwarového kódu ve vývojovém prostředí. Průběžné dodání a nasazení přenáší tuto integraci na další úroveň, a to kombinací rychlého a bezpečného nasazení.

### Průběžná integrace (Continuous Integration)

Postup, ve kterém je software vyvíjen a integrován různými členy týmu v jednom balíku do shodného úložiště. Integrace probíhá obvykle alespoň jednou denně a každá z integrací je zkontrolována automatickým sestavením, kde se zjistí případné integrační chyby. [24]

### Průběžné dodání (Continuous Delivery)

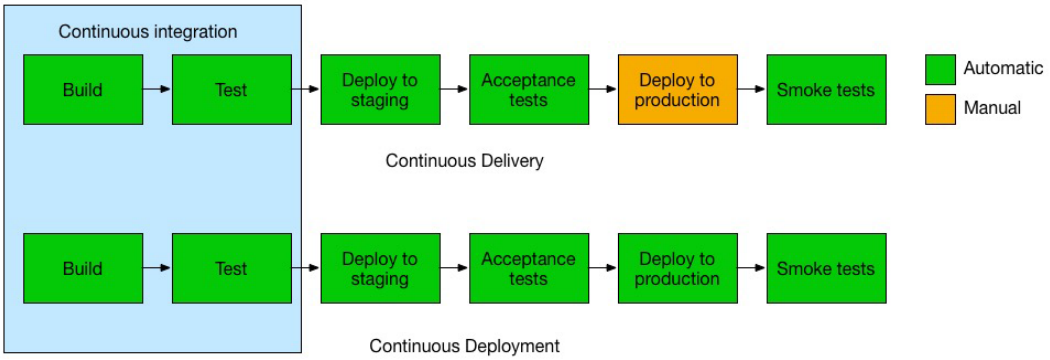
Je proces, ve kterém je software průběžně dodáván do určitého prostředí např. do testovacího nebo produkčního, kde je poté využíván skutečnými uživateli. Průběžné dodání využívá průběžnou integraci a určitým způsobem lze Continuous Delivery považovat za vyšší formu Continuous Integration. To tedy znamená, že před finálním nasazením projde software procesem kontroly v rámci průběžné integrace. Průběžné dodání není zaměřeno pouze na nasazení aplikace do určitého prostředí, ale zahrnuje celkovou změnu pohledu na vývoj aplikace, kdy každé úspěšné sestavení může být nasazené do produkčního prostředí na vyžádání. [24]

### Průběžné nasazení (Continuous Deployment)

Průběžné neboli nepřetržité nasazení softwaru jde ještě o krok dále než průběžné dodání. To v praxi znamená, že jakákoliv změna, respektive každá nová verze aplikace

automaticky projde všechny fáze nasazení až do produkčního prostředí, kde je přístupná koncovým uživatelům. Do procesu nasazení nikdo manuálně nezasahuje a pouze selhání automatického průběhu zabrání novému doručení aplikace do produkce. [24]

Jak je možné vidět na obrázku níže, průběžná integrace je součástí průběžného dodání i nasazení. A průběžné dodání a nasazení se liší jen v nasazení do produkčního prostředí, které je v Continuous Delivery nutné vykonat manuálně a v Continuous Deployment se uskuteční automaticky.




**Obrázek 3.1:** Continuous Integration vs. Continuous Delivery vs. Continuous Deployment (převzato z [24])

Velmi užitečným nástrojem pro průběžné dodání a nasazení softwaru je již zmíněný Docker, kde díky oddělení kontejneru od prostředí, ve kterém je aplikace spuštěna, jsou minimalizovány chyby spojené s rozdílností různých prostředí.

Dalším důležitým faktorem pro automatické nasazení aplikací je orchestrace. Orchestrace je proces integrace dvou a více aplikací nebo služeb do jednoho automatizovaného procesu nebo synchronizace jejich dat v reálném čase.

## 3.1 Orchestrace mikroslužeb

Při vytváření aplikací prostřednictvím softwarových kontejnerů se může pracovat až se stovkami kontejnerů, které je potřeba vhodným nástrojem ovládat. Proto existují různé nástroje, tzv. orchestrátory, které slouží ke správě a ovládání kontejnerů, a tím usnadňují práci s kontejnery.

Orchestrátory rozdělují jednotlivé kontejnery do logických skupin a poté pracují

se skupinou samostatně. Díku tomu lze s logickou skupinou pracovat jako s jedním kontejnerem. To znamená, že případné změny se definují pro skupinu a orchestrátor změny vykoná ve všech kontejnerech.

Pomocí orchestrace lze vzít jednotlivé aplikace v kontejnerech a rozdělit je přes vícenodové infrastrukturní zdroje, řídit jejich umístění, škálování, vyvažování zátěže (load balancing) a další. [25]

Cloud-native platforma, která ještě může být spuštěna nad orchestrátorem, zajišťuje vývoj a provoz mikroslužeb. Zajistí i realizaci kontejneru, tedy neočekává jen předložení hotového image, ale poskytne zdrojový kód aplikace a sestaví potřebné prostředí, framework a balíčky, které poté udržuje v provozu. Tento způsob je nejrychlejší, jak dostat aplikaci do světa cloudu a naplno tak využít možnosti agilního vývoje. [6]

Vzhledem k faktu, že Docker je poměrně mladý projekt, vzniká mnoho nástrojů pro jeho orchestraci, ale s různým stupněm kvality. Převážně se liší ovladatelností a ne všechny vyhovují produkčnímu prostředí či použití ve velkých infrastrukturách. U jednotlivých nástrojů je také rozhodujícím faktorem počet kontejnerů, se kterými orchestrátor dokáže pracovat. [25]

### 3.1.1 Docker Compose

Docker Compose je nejprimitivnější nástroj pro práci s kontejnery a slouží pro orchestraci jednoho nebo i více kontejnerů na jednom počítači. Primárně je určen především ke spouštění více kontejnerových aplikací a na rozdíl od ostatních nabízí relativně málo funkcí. Používá soubor "Compose" ke konfiguraci všech služeb dané aplikace, které poté lze spustit pomocí jediného příkazu - *docker-compose up*.

Hlavním myšlenkou Docker Compose je tedy propojit několik existujících kontejnerů do jednoho společného, což umožní vytvořit komplexní aplikaci z několika image. Dále Docker Compose automatizuje nasazování a konfiguraci kontejnerů. Bez tohoto nástroje by bylo nutné spustit příkaz *docker run* zvlášť pro každý kontejner se všemi požadovanými parametry, čímž by se spouštění stalo značně náročné a zdlouhavé.

Jednoduchost tohoto nástroje je i jeho nevýhodou, protože nenabízí skoro žádné pokročilé funkce. Je vhodný zejména pro samostatné vývojáře a menší vývojové týmy, které potřebují pouze jednoduché vývojové prostředí s předem definovanými službami. [26]

### 3.1.2 Docker Swarm

Je nástroj přímo od společnosti, která stojí za technologií Docker a umožňuje ze skupiny kontejnerů vytvořit cluster, na kterém běží Docker Engine.

Je to v podstatě také jednoduchý nástroj, a to především z toho důvodu, že API pro Docker Swarm je prakticky stejné jako API pro Docker. Tudíž jsou dostupné všechny příkazy z Docker Engine a ještě i některé navíc pro nastavení swarmu a rozmístění kontejnerů.

Všechny Docker Engine, které jsou součástí daného clusteru, jsou spuštěné v tzv. swarm režimu. Tento režim je možné vytvořit samostatně nebo připojením enginu k již existujícímu swarmu. Samotný swarm představuje více vzájemně propojených Docker Enginů, na které lze nasadit jednotlivé služby.

Docker Swarm pomocí integrace na ostatní nástroje Dockeru dokáže vytvořit požadovaný cluster ve velmi krátkém čase. Díky stejnému API lze v příkazovém řádku celý nový cluster ovládat velmi podobně jako jeden samotný Docker. Nevýhodou je, že dokáže pracovat pouze s Docker kontejnery. [27]

### 3.1.3 Apache Messos

Je projekt, který vznikl před vznikem Dockeru a ostatních orchestračních nástrojů již v roce 2009 a také je poněkud odlišný, a to proto, že Apache Messos je dvouúrovňový scheduler a není tak zaměřen pouze na kontejnery.

První úroveň má na starost správu zdrojů a přiděluje zdroje frameworkům. Druhá úroveň řeší určité plánování v rámci frameworku. Na vytváření a spouštění kontejnerů pak slouží framework s názvem Marathon.

Apache Messos spojuje více věcí do jediného systému, přičemž orchestrace kontejnerů je jen jednou z jeho částí. Celý nástroj je ale ověřen ve velkých instalacích a má poměrně vysokou spolehlivost. [28]

### 3.1.4 Kubernetes

Projekt Kubernetes byl vytvořen a stále je z velké části vyvíjen a spravován společností Google. Jde o plnohodnotnou platformu pro správu Docker kontejnerů. Využívá se v mnoha produkčních prostředích a je vhodný i pro tvorbu hybridního clusteru v rámci několika

různých poskytovatelů.

Kubernetes je open-source systém, který podporuje automatické nasazení, škálování a správu kontejnerových aplikací. Je navržen podle ověřených principů, díky kterým Google provozuje více jak miliardu kontejnerů. Google tak navrhl open-source platformu, která je určena pro použití i u ostatních, konkurenčních poskytovatelů cloudu AWS a Microsoft Azure. Kubernetes dovoluje vývojářům nasadit aplikace do libovolné cloudové platformy, aniž by museli významně modifikovat aplikaci.

Hlavní výhodou Kubernetes je samostatné nasazování kontejnerů, kdy se sleduje jejich dostupnost a efektivně se využívá veškerá výpočetní kapacita, která je k dispozici. To umožňuje udržet aplikaci co nejdostupnější. Kubernetes vytváří abstrakci nad dostupným hardwarem a všechny připojené servery se tak chovají jako jeden stroj. [29]

Podobně jako Apache Mesos si tak řadí všechny fyzické zdroje do jednoho celku, ze kterého si podle potřeby postupně rozděluje zdroje. Avšak na rozdíl od ostatních orchestrátorů nabízí mnoho funkcí, je snadno rozšiřitelný a podporuje více kontejnerových řešení - Docker, Rocket, Windows kontejnery a další.

Kubernetes seskupuje kontejnery do logických celků tvz. Podů. Jedná se o seskupení kontejnerů, které sdílí společné zdroje – nejčastěji je to síť a datové úložiště. Kubernetes poskytuje velké množství nastavení a je vhodný i pro velké projekty a datová centra. [29]

### Pod

V Kubernetes jsou všechny kontejnery provozovány v Podech. Pod je základní komponenta, která se stará o plánování jednotlivých činností. Skládá se z jednoho nebo i více kontejnerů, které logicky patří k sobě, mají sdílené úložiště nebo síť a specifikace, jak spouštět kontejnery (například aplikační server a proxy). Lze tedy více kontejnerů spravovat jako jednu společnou aplikaci. Toto spojení umožňuje, že jednotlivé kontejnery, které k sobě náleží, budou spuštěny společně na stejném počítači. Kontejnery patřící do stejného Podu sdílejí identické prostředí, můžou komunikovat se stejnými zařízeními a využívají totožné IP rozsahy. Škálování je také prováděno s Pody jako celky.

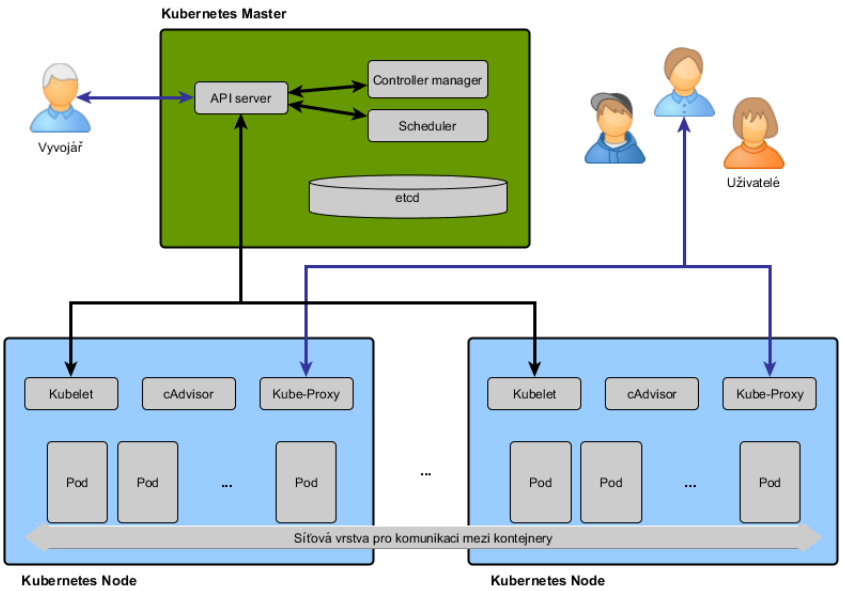
Uvnitř Podu se zpravidla nachází jeden ústřední kontejner a několik pracovních kontejnerů. Hlavní kontejner, často nazývaný jako manažerský, poskytuje veškerou funkcionalitu a přerozděluje jednotlivé úlohy na pracovní kontejnery. [29]

Dle výše zmíněného lze Pod přirovnat k virtuálnímu počítači, který dokáže úspěšně přerozdělovat přidělené úlohy.

### Architektura

Kubernetes architektura se skládá z Master a Node komponent.

Kubernetes Master obsahuje složky, které spravují a řídí Kubernetes Nody a rozdělují běžící kontejnery. Master je jediným bodem selhání, jelikož v případě selhání ztrácí funkčnost. Aby se zabránilo těmto poruchám, je možné spustit vice Masterů najednou a zajistit tak vysokou dostupnost. [30]




**Obrázek 3.2:** Kubernetes - architektura (převzato z [31])

Master zahrnuje:

- Etcd - je distribuované a konzistentní úložiště klíčových hodnot pro sdílenou konfiguraci a zajišťování služeb všech REST API objektů. Mělo by být nakonfigurované pro vysokou dostupnost, poněvadž je to jediné místo, kde je ukládán aktuální stav. Etcd také slouží ke sledování změn a jsou zde uložena veškerá data z clusteru.
- Controller Manager - spouští ostatní controllery jako podprocesy na pozadí, které se zabývají rutinními úlohami v rámci clusteru. Logicky je každý controller samostatným procesem, ale pro snížení složitosti jsou všechny sestaveny do jediného binárního souboru a běží jako jeden proces. Controller Manager je řídící smyčkou, která sleduje

sdílený stav clusteru přes API server a provádí změny, při nichž se snaží o přesun z aktuálního stavu do požadovaného.

- API server - ověřuje a konfiguruje data pro API objekty, které zahrnují – Pody, služby, replikační controller a další. Dále poskytuje REST (Representational State Transfer) operace, které jsou navržené pro distribuované prostředí, ale neprovádí žádné akce, pouze udržuje stav.
- Scheduler (plánovač) - dohlíží na nově vytvořené Pody, které zatím nemají přiřazený Node a následně pro tyto Pody vybírá vhodný Node, kde budou spuštěny. Umožňuje tak velmi efektivně provozovat kontejnery a významně ovlivňuje dostupnost, výkon a kapacitu. [30]

Kubernetes Node neboli uzel je označení pro pracovní stroj. V závislosti na clusteru může být Node virtuální nebo i fyzický stroj. Každý Node obsahuje služby nezbytné ke spuštění Podů a aplikačních kontejnerů. Nody jsou spravovány a řízeny přes - Kubernetes Master. [29]

Mezi služby, které jsou na každém Nodu, patří:

- Kubelet - je nejdůležitější a nejvýznamnější controller. Působí jako most mezi Kubernetes Master a Nody a přímo komunikuje s API serverem. Kubelet řídí všechny Pody a kontejnery, které na zařízení běží a načítá statistiky jednotlivých kontejnerů z cAdvisor. Dále zajišťuje spouštění a zastavování kontejnerů a spravuje úložiště.
- Kube-proxy - na hostitelském systému dovoluje abstrakci objektů, které jsou typu služba, správou síťových pravidel. Dále předává požadavky vyhovujícím kontejnerům, zajišťuje balancování a zodpovídá za předvídatelné, izolované a přístupné síťové prostředí. Kube-proxy běží na každém Nodu a odráží služby definované v API rozhraní.
- cAdvisor - je agent pro analýzu výkonu. V Kubernetes je cAdvisor integrován přímo do binárního systému Kubelet, kde monitoruje a shromažďuje využití prostředků a metriky výkonu na každém nodu. Agent cAdvisor automaticky vyhledá všechny kontejnery v daném zařízení a sleduje jejich statistiky - využití procesoru, paměti, souborového systému a sítě. [30]

## 3.2 Řešení pro multi-cloud delivery

V dnešním světě cloud computingu probíhá aktivní snaha provozovat řešení doručování aplikací do prostředí multi-cloudu. Řešení, které zajistí, že aplikace jsou jednoduše a spolehlivě dodávané uživatelům bez ohledu na to, kde se nacházejí a jak je potřebují. Proto existují specifické nástroje, které doručování aplikací do multi-cloudu podporují.

### 3.2.1 Cloud Foundry

Je open-source multi-cloud služba typu PaaS, kterou spravuje firma Pivotal (nezávislý subjekt financovaný firmou VMware). Platforma slouží pro vytváření, nasazování, řízení a provozování aplikací vyvíjených v různých jazycích a architekturách. Byla vytvořena za účelem zjednodušení a zrychlení procesu vývoje, nasazení a práce s aplikacemi.

Cloud Foundry využívá výhod multi-cloudu, tudíž je možno jej provozovat v rámci mnoha cloudů a cloudových infrastruktur a vývojáři se nemusí omezovat pouze jedním konkrétním cloudem. Tato vize se stala skutečností především díky širokému spektru soukromých cloudů, správy cloudu, řešení pro nasazení a poskytovatelů veřejného cloudu, kteří Cloud Foundry podporují. [32]

Na této platformě lze tedy spustit a provozovat téměř každou aplikaci. Je ale zapotřebí vytvořit sestavovací konfiguraci a následně v cloudovém prostředí aplikaci sestavit. Kompletní konfigurace je poměrně složitá, avšak je k dispozici již mnoho předpřipravených řešení, které je možné využít.

Aplikace provozované na platformě Cloud Foundry jsou nasazovány, škálovány a spravovány prostřednictvím open source nástrojů BOSH, které slouží pro nasazování a správu životního cyklu rozsáhlých distribuovaných služeb. BOSH je v podstatě orchestračním nástrojem běžícím na vlastní virtuální instanci a ostatní instance nasazuje a spravuje. K tomuto účelu na každou instanci, kterou nasadí, přidává BOSH Agenta, přes kterého instance ovládá a monitoruje. [33]

### 3.2.2 Spinnaker

Open-source projekt Spinnaker je multi-cloud platformou pro continuous delivery, která slouží k uvolňování nových verzí softwaru s vysokou rychlostí a spolehlivostí. Spin-

naker byl primárně vytvořen firmou Netflix pro Amazon Web Services, ale podporuje i další poskytovatele - Google Cloud Platform, Kubernetes, OpenStack, DC/OS a Microsoft Azure. Nyní je Spinnaker považován za jeden z nejlepších nástrojů pro automatizaci nasazení aplikací do multi-cloud. Využívá jej např. americká banka Target, GPS navigace Waze, cloudové úložiště Box a další.

Nástroj Spinnaker pomáhá s nasazením aplikací do prostředí cloudu nebo i multi-cloudu. Cílem je vytvoření a nasazení nezměnitelných image na několik různých poskytovatelů pomocí různých strategií, které jsou součástí Spinnakeru. [34]

Spinnaker poskytuje dvě hlavní funkce - správu clusterů a správu nasazení. Správa clusterů se týká infrastruktury aplikace a správa nasazení řídí proces nasazení aplikace. Spinnaker není typu PaaS, jak by se mohlo na první pohled zdát, ale je určen pro velkou orchestraci aplikací.

Spinnaker nabízí uživatelské rozhraní dostupné přes webový prohlížeč, které umožňuje vytvářet a spravovat aplikace, clustery, pipeline, load balancery, skupiny zabezpečení a skupiny serverů tvořené virtuálními počítači a kontejnery. [34]

Proces průběžného dodání začíná určením cílových platforem pro nasazení - virtuální počítač, PaaS nebo kontejnerové orchestrační nástroje (např. Kubernetes). Poté následuje definování pipeline, které mohou obsahovat několik fází nasazení, a to do jedné nebo i více cílových platforem.

Pipeline lze vnímat jako schéma projektu - sekvence jednotlivých fází, kterými se řídí nasazení aplikací konzistentně, opakovaně a bezpečně. Pipeline vždy začíná konfigurační fází, ve které se nastaví spouštěč (Trigger), jenž definuje, po jaké akci se pipeline spustí. Touto akcí může být odeslání kódu do Git repozitáře, naplánovaná událost v cron, nahrání nového Docker image do Docker Registry nebo úspěšné dokončení předchozí pipeline. Další podstatnou fází, kterou obsahuje v podstatě každá pipeline, je nasazení - to zahrnuje především výběr image, který se má nasadit a volbu strategie nasazení. Neměnný pipeline spolehlivě a důsledně dodržuje nepřetržité dodávání softwaru. [35]

Jakmile je vybrané prostředí pro nasazení Spinnakerem registrováno, vytvoří se následující zdroje, které mapují přesné protějšky v cílovém prostředí:

- Load balancer - vyrovnává datový provoz mezi instance ve skupinách serverů, je to tedy koncový bod, kde bude aplikace po nasazení dostupná. Obvykle je k němu přiřazena veřejná IP adresa, tzv. Ingress, aby byla aplikace přístupná prostřednictvím

internetu.

- Skupiny zabezpečení - sada pravidel ve firewallu, které jsou přeloženy do síťové politiky v příslušném prostředí.
- Cluster - definuje sadu výpočetních prostředků spojených s nasazením. Typicky představuje izolované prostředí, jako například vývojové, testovací nebo produkční. Každý z clusterů může obsahovat jednu nebo více skupin serverů a lze ho nakonfigurovat tak, aby nastavil požadovaný stav libovolné skupině serverů.
- Skupina serverů - je elastická součást nasazení, která se skládá z výpočetních prostředků, jako jsou virtuální počítače, kontejnery nebo Pody, a je řízená skupinou zabezpečení. Každá skupina serverů má minimální a maximální počet výpočetních zdrojů, které přesně stanovují požadovaný stav. [36]

### Architektura

Spinnaker se skládá z několika nezávislých mikroslužeb, z nichž zásadní jsou:

- Deck - uživatelské rozhraní dostupné přes webový prohlížeč. Zobrazuje stav a historii spuštěných pipeline.
- Gate - API brána. Uživatelské rozhraní a všechny služby komunikující s API jsou spojeny se Spinnakerem přes tuto bránu.
- Orca - orchestrátor, který spravuje, řídí a je odpovědný za konkrétní operace a pipeline.
- Clouddriver - je zodpovědný za přeměnu volání cloudovým poskytovatelům a za indexování a ukládání všech nasazených zdrojů.
- Echo - slouží k odesílání notifikací, jako např. e-mail, SMS, Slack a další.
- Fiat - je autorizační služba Spinnakeru. Používá se k řízení přístupu a oprávnění k uživatelským účtům a aplikacím.
- Front50 - úložiště, které uchovává metadata vytvořených projektů, aplikací, pipeline a oznámení.
- Halyard - je konfigurační služba neboli deamon. Řídí životní cyklus všech ostatních mikroslužeb a je s ostatními v interakci v případě spouštění, aktualizace nebo návratu k předchozí verzi nasazované aplikace. [37]

### Strategie nasazení

Spinnaker obsahuje různé strategie, kterými lze aplikace nasadit. Jde především o to, jestli budeme vyžadovat zálohování jednotlivých nasazení nebo stačí pouze aktuálně nasazená verze.

- Canary - Spinnaker nasadí dvojici clusterů, které se skládají ze stávající verze aplikace a nově vydané verze. Malé procento datového provozu je poté přesměrováno na nový cluster, a tím je umožněno provádět analýzu nové verze v reálném čase.
- Red/Black - nasadí novou skupinu serverů, přidá ji k load balanceru a směruje na ni všechen datový provoz. Nabízí výběr, co bude provedeno se starší skupinou serverů, může být zmenšena nebo vypnuta. Pokud nedojde k odstranění, vždy lze jakoukoli skupinu serverů obnovit.
- Rolling Red/Black - je kombinací předchozích strategií Canary a Red/Black. Nová skupina serverů je umístěna vedle předchozí. Analýza nové verze se provádí průběžně a při postupu se paralelně vyřazuje starší verze.
- Highlander - tato strategie vytvoří novou skupinu serveru, kterou také přidá k load balanceru. Rozdíl oproti předchozí strategii je, že jakmile je nová skupina úspěšně vytvořena, předešlé skupiny serverů budou zničeny. Nelze tedy uskutečnit návrat do předchozího stavu.
- Blue/Green - v této strategii je nasazena nová skupina serverů vedle předchozí s identickým počtem instancí. Až po otestování, že nová verze splňuje všechny požadavky, load balancer přepne provoz ze starší verze na novou. Nevýhodou je, že strategie vyžaduje více výpočetních zdrojů (instancí).
- None - pouze nasadí novou skupinu serverů, ale nebude dělat nic se staršími skupinami. Všechny skupiny serverů budou aktivní v load balanceru. [36] [38]

## 3.3 Kontejner jako služba - CaaS

Je model kontejnerové virtualizace, ve kterém jsou kontejnery, orchestrace a podpůrné výpočetní prostředky dodávány uživatelům jako služba. Díky CaaS mohou uživatelé nahrávat, organizovat, spouštět, spravovat a zastavovat kontejnery pomocí volání API nebo přes webové rozhraní poskytovatele. Stejně jako v případě ostatních modelů, i zde uživatelé

platí pouze za prostředky, které skutečně využijí. [39]

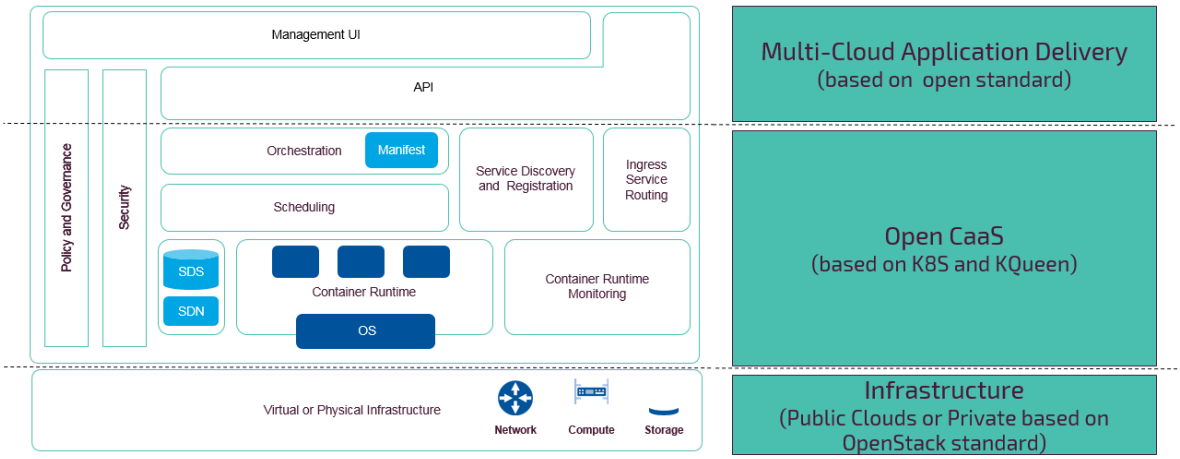
V rámci struktury distribučních modelů cloud computingu se kontejner jako služba nachází mezi modely infrastruktura jako služba a platforma jako služba. Nicméně CaaS je nejčastěji brán jako podmnožina IaaS, kde je základním zdrojem kontejner.

Klíčové komponenty CaaS podle Gartner[40] lze rozdělit do 3 vrstev:

Vrchní vrstva zahrnuje uživatelské rozhraní a API. Tato vrstva historicky spolupracuje s vrstvami níže, čímž dovoluje vývojářům volat API rozhraní a poskytovat určitou službu v PaaS. Ale v multi-cloud delivery se tato vrstva otočí a začne vyhledávat v aplikaci, což vývojářům umožní definovat, jak bude jejich aplikace vypadat.

Prostřední vrstva, která se v minulosti skládala z mnoha proprietárních standardů (např. kontejnerové knihovny, síťování, monitorování a další), je nyní rozdělena na tzv. otevřené komponenty. Multi-cloud strategie je zaměřena na poskytování podpory všem těmto komponentám, ale obecně umožňuje vývojářům a uživatelům zvolit si, která z nich je pro jejich aplikaci nejvýhodnější.

Spodní vrstva se skládá z fyzické nebo virtuální infrastruktury. Patří sem síťové prvky, počítače, datová úložiště a další, které jsou umístěné ve veřejném nebo privátním cloudu založeném například na technologii OpenStack.




**Obrázek 3.3:** CaaS - komponenty

# 4 Webová aplikace

Cílem této práce je efektivně doručit a nasadit webovou aplikaci do prostředí multi-cloudu. Pro vysvětlení konceptu webové aplikace se nejprve musíme zaměřit na vymezení pojmu webové stránky. To jsou stránky s obsahem napsaném v jazyce HTML (HyperText Markup Language), které jsou vzájemně propojeny pomocí hypertextových odkazů.

Webová aplikace je složena ze statických a dynamických webových stránek. Statická webová stránka je neměnná. Jestliže si ji uživatel vyžádá - webový server odešle stránku webovému prohlížeči, který o ni požádal, bez jakékoliv změny. Naproti tomu dynamickou webovou stránku server upraví před odesláním prohlížeči, který o ni požádal. Proměnlivá povaha stránky je důvodem, proč se nazývá dynamická. [41]

Statické webové stránky se skládají z jedné nebo více souvisejících HTML stránek a souborů uložených v zařízení (fyzickém nebo virtuálním), na kterém běží webový server. Webový server je software, který odesílá webové stránky na základě požadavků od webových prohlížečů. Konečný obsah statické webové stránky je tedy určen HTML kódem a obsah je neměnný.

V případě, že webový server přijme požadavek na dynamickou stránku, nepošle ji ihned webovému prohlížeči, který o ni požádal jako v případě statické stránky, ale předá žádost zvláštnímu softwaru, který se nazývá aplikační server, jenž je zodpovědný za dokončení stránky. Aplikační server postupně zpracuje kód na stránce, čímž dokončí stránku dle instrukcí v kódu a nakonec dynamický kód ze stránky odstraní. Výsledkem je statická stránka, kterou aplikační server předá zpět webovému serveru a ten ji přepošle prohlížeči, jenž o ni požádal. Prohlížeč tedy znovu dostane statický HTML kód, který zobrazí. [41]

Výhody webových aplikací:

- Nemusí se instalovat lokálně.
- Uživatel nemusí aplikace aktualizovat (aktualizace probíhá na serveru).
- K využívání je nutný pouze webový prohlížeč.

- Jsou uložené na vzdáleném serveru a přístupné skrz internet odkudkoliv.

  Nevýhody webových aplikací:

- Vyžadují stálé připojení na internet.
- Může nastat pomalejší tok dat v závislosti na kvalitě a rychlosti připojení.
- Možná bezpečnostní rizika úniku dat v případě špatného zabezpečení poskytovatele. [41]

Vývoj webové aplikace zpravidla probíhá na lokálním zařízení, případně v testovacím prostředí, které není přístupné koncovým uživatelům. Po dokončení vývoje je tedy nutné aplikaci doručit na veřejně dostupný server, na který se mohou uživatelé připojit a s aplikací komunikovat. Tímto se rozumí proces nasazení a neobejde se bez ní žádná webová aplikace.

## 4.1 Představení aplikace

V teoretické části byly představeny nástroje, které budou využity pro implementaci webové aplikace. Výsledná aplikace nebude disponovat velkým množstvím funkcí, jedná se pouze o příklad použití webové aplikace a nástrojů nutných k automatickému průběžnému dodání softwaru do prostředí multi-cloudu.

V této práci bude použita webová aplikace postavená na webovém serveru Nginx v Docker kontejneru. Nginx je open source softwarový web server, který pracuje s protokoly HTTP, HTTPS, SMTP, POP3, IMAP a SSL. Nginx nabízí vysokou výkonnost a stabilitu, jednoduchou konfiguraci, mnoho funkcí a zároveň nízkou spotřebu zdrojů. [42]

V případě nasazení kontejnerové aplikace do produkčního prostředí je třeba vymyslet mechanismus, jakým přepnout ze staré verze aplikace na novou. V případě samotného Dockeru je to řešeno poměrně obtížně - je třeba nakonfigurovat proxy, který nasměruje provoz na správný kontejner, poté spustit nový, následně přepnout proxy a nakonec starý kontejner odstranit. Snadnější varianta bez konfigurace proxy zahrnuje krátký nežádoucí výpadek aplikace. Proto budou použity nástroje Kubernetes a Spinnaker, díky kterým lze snadno nasadit aplikaci bez výpadku.

Počáteční nahrání a následné úpravy zdrojového kódu budou prováděny přes nejrozšířenější verzovací nástroj GitHub, který obsahuje open-source systém Git. Git vyvi-

nul autor Linuxu Linus Torvalds pro potřeby vývoje Linuxového jádra. Jeho předností je vlastní, stejně pojmenovaný protokol určený pro přenos dat, který bezpečně komunikuje díky SSH (Secure Shell). Ve srovnání s ostatními verzovacími systémy nabízí Git rozsáhlou škálu příkazů, přesto základní práce s repozitáři není složitější než je tomu u jiných systémů. Velkou výhodou je také jeho hlavní webová služba GitHub, která se významně podílí na jeho popularizaci tím, že uživatelům poskytuje přívětivé grafické rozhraní pro správu jejich příspěvků k repozitářům a i díky tomu kolem Gitu vznikla velmi rozsáhlá komunita vývojářů. [43]

Pro prvotní nasazení bude využita statická HTML stránka - "Hello World", která ověří funkčnost celého navrhovaného řešení. Zdrojový kód vytvoříme v GitHub, kde založíme nový repozitář s názvem nginx-multicloud, kam vložíme skript Dockerfile a složku html se souborem index.html.

V Dockerfile je v části FROM definováno, z jakého kontejneru chceme vycházet a jaký webový server se použije. Akce COPY vyvolá kopírování zdrojového HTML kódu do výchozího adresáře webového serveru:

```
FROM nginx:alpine

COPY /html /usr/share/nginx/html
```

**Ukázka kódu 4.1:** Dockerfile pro vytvoření webové aplikace

HTML kód v index.html:

```
<html>
  <header><title>This is title</title></header>
    <body>
      <h1>Hello world - Spinnaker</h1>
    </body>
</html>
```

**Ukázka kódu 4.2:** HTML kód - Hello World

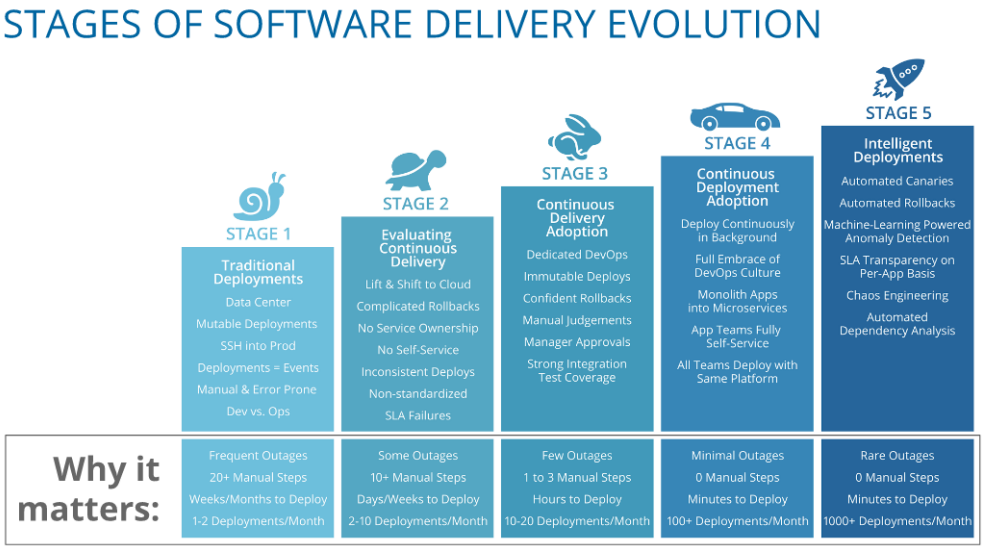
Poté budou prováděny změny v aplikaci a výsledná struktura HTML stránky bude zahrnovat:

- Nadpis
- Logo Spinnaker
- Text s informacemi o nasazení a aplikaci
- Aktuální datum a čas

- IP adresu webového serveru, na kterém aplikace běží

## 4.2 Definice měření

Rozlišuje se 5 etap implementace softwaru - to pomáhá identifikovat, na jaké úrovni automatizovaného dodávání nebo nasazení softwaru dané řešení je.




**Obrázek 4.1:** Etapy vývoje softwaru [44]

1. Tradiční způsob nasazení - zahrnuje velké množství manuálních zásahů uživatele a s tím spojené časté výpadky aplikace. Nasazení probíhá velmi dlouho, tudíž je možné za měsíc nasadit jednu až dvě aplikace.
2. Poznání průběžného doručení - v této fázi je počet zásahů od uživatele do průběhu nasazení stanoven na 10 a více, doba nasazení je ve dnech až týdnech, ale dochází k méně výpadkům. Problémem je složitost návratu do předchozí verze aplikace. Jedná se o fázi, ve které se aplikace přesunují do cloudu a díky tomu je nasazení rychlejší.
3. Průběžné doručení - zásluhou využití průběžného dodání je méně vyžadován manuální zásah uživatele, samotné nasazení je otázkou několika hodin, a tak lze nasadit až 20 aplikací za měsíc.
4. Průběžné nasazení - nasazení probíhá průběžně jako proces v pozadí. Tudíž není potřeba žádný zásah od uživatele, dochází pouze k minimálním výpadkům aplikace a nasazení je velmi rychlé (v řádu minut). Díky tomu se za měsíc nasadí více jak 100 aplikací. Tato etapa zahrnuje proces transformace monolitických aplikací na mikroslužby.

5. Inteligentní nasazení - umožňuje nasadit přes 1000 aplikací měsíčně a je velmi podobné předchozí etapě. Avšak výpadky aplikace jsou vzácností a návrat k předchozí verzi je automatizovaný. Vše tedy probíhá automaticky a velmi rychle. [44]

Níže je definováno jakým způsobem, a jakými ukazateli bude probíhat měření při automatickém dodávání aplikace do prostředí multi-cloudu.

Klíčové ukazatele:

- Čas do nasazení v produkčním prostředí - jak dlouhá doba uplyne od dokončení kódu aplikace až do jejího finálního nasazení do produkčního prostředí.
- Počet uživatelských zásahů - kolik kroků musí uživatel při nasazení aplikace provést manuálně. Uživatelé si často ani neuvědomují, kolik manuálních zásahů musejí v celém průběhu nasazení dané aplikace vykonat.
- Návrat k předchozí verzi - ve většině případů není možný návrat k předchozí verzi aplikace a v případě, že návrat možný je, tak pouze pomocí složitých procesů, které jsou náchylné k chybám.
- Počet použitých nástrojů - určuje, kolik nástrojů je potřeba využít v průběhu celého procesu nasazování aplikace.

# 5 Návrh a implementace aplikace do prostředí multi-cloud

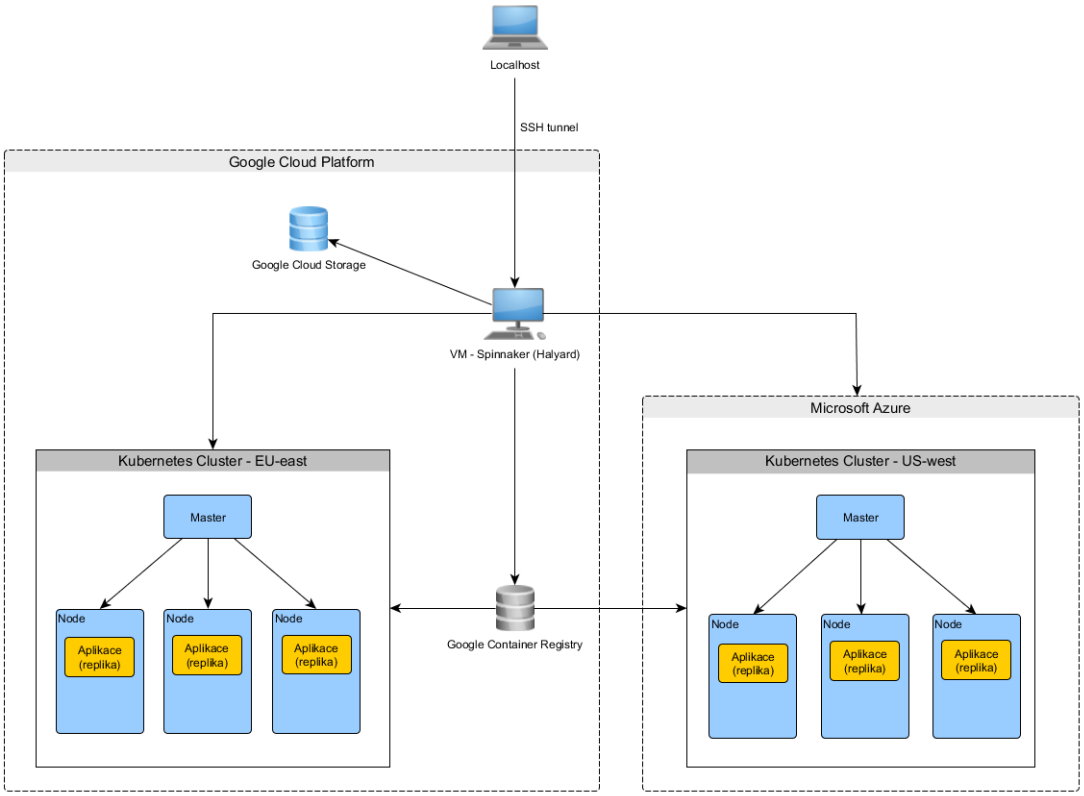
Aplikace bude nasazena na dva Kubernetes clustery - jeden na Google Cloud Platform a druhý na Microsoft Azure. Tito dva poskytovatelé byli vybráni z důvodu, že v Google Cloud má Kubernetes nativní podporu a umožňuje snadné nasazení clusteru prostřednictvím webového rozhraní a Microsoft Azure nabízí také poměrně jednoduchou možnost nasazení pomocí šablony. Clustery jsou tak spuštěny během chvíle bez nutnosti manuální instalace a není třeba se starat o provoz Master nodu nebo aktualizace. Vše je snadno propojitelné se zbytkem cloudu (úložiště, load balancery) a snadno lze nastavit automatické škálování a další náležitosti.

Aspektem, který je také třeba zvážit, je cena za provoz clusteru. Pokud je Kubernetes provozováno na vlastním hardwaru, nic se nemění, samotný Kubernetes je zdarma. Ale ve chvíli, kdy se přejde do cloudu, je cena za provoz serverů podstatně vyšší. Výhodou vybraných poskytovatelů je, že je lze využívat bezplatně - Google Cloud Platform nabízí každému, kdo se zaregistruje vstupní kredit 300 dolarů, který může čerpat po dobu jednoho roku. V Microsoft Azure je možné založit studentský účet, u kterého je k dispozici 100 dolarů.

V Google Cloud Platform bude virtuální počítač s veřejnou IP adresou, na kterém bude nainstalován Spinnaker, který vhodně doplní Kubernetes pomocí pipeline umožňující zvolit více strategií nasazení. Dále zde bude úložiště perzistentních dat a Google Container Registry pro vytváření a ukládání jednotlivých image. K tomu bude použit nástroj Docker, a to z důvodu zaměření na nasazování malých kontejnerů, které se v případě aktualizace snadno a rychle vymění za nové.

Spinnaker byl vybrán protože spolupracuje s nástroji pro průběžnou integraci, podporuje nasazení na všechny hlavní veřejné poskytovatele cloudu a užívá osvědčené postupy pro nasazování softwaru (všechny implementace jsou prováděny prostřednictvím nezměnitelných image).

V Google Cloud Platform i v Microsoft Azure bude vytvořen Kubernetes cluster s jedním Master a třemi Node komponentami, přičemž každý z clusterů v jiné geografické lokalitě.




**Obrázek 5.1:** Schéma multi-cloud prostředí

## 5.1 Příprava multi-cloud prostředí

V následující podkapitole je popsána příprava multi-cloud prostředí s vytvořením dvou Kubernetes clusterů v Google Cloud Platform a Microsoft Azure.

### 5.1.1 Google Cloud Platform

Nejprve je nutné vytvořit nový projekt v prostředí Google Cloud Platform, ve kterém povolíme přístup k těmto API rozhraním - Google Identity and Access Management (IAM) API, Google Cloud Resource Manager API a Google Container Registry API.

Poté je důležité založit účet služby, který se používá k identifikaci a povolení přístupu nástrojům spuštěným na virtuálním počítači do ostatních služeb Google Cloud Platfrom. Tento účet vytvoříme v příkazovém řádku Google Cloud Shell, který je dostupný ve webovém rozhraní, následujícími příkazy (proměnná GCP_PROJECT je název projektu

a HALYARD_SA název vytvářeného účtu):

```
GCP_PROJECT=my-spinnaker
HALYARD_SA=halyard-service-account

gcloud iam service-accounts create $HALYARD_SA \
    --project=$GCP_PROJECT \
    --display-name $HALYARD_SA

HALYARD_SA_EMAIL=$(gcloud iam service-accounts list \
    --project=$GCP_PROJECT \
    --filter="displayName:$HALYARD_SA" \
    --format='value(email)')

gcloud projects add-iam-policy-binding $GCP_PROJECT \
    --role roles/iam.serviceAccountKeyAdmin \
    --member serviceAccount:$HALYARD_SA_EMAIL

gcloud projects add-iam-policy-binding $GCP_PROJECT \
    --role roles/container.admin \
    --member serviceAccount:$HALYARD_SA_EMAIL
```

**Ukázka kódu 5.1:** Vytvoření účtu služby - halyard-service-account

Druhý účet služby s názvem *gcs-service-accout* bude použit v další části této práce při nastavení perzistentního úložiště dat pro Spinnaker. Opět je tedy nutné povolit přístup do ostatních služeb Google Cloud Platfrom, především do Google Cloud Storage:

```
GCS_SA=gcs-service-account
gcloud iam service-accounts create $GCS_SA \
    --project=$GCP_PROJECT \
    --display-name $GCS_SA

GCS_SA_EMAIL=$(gcloud iam service-accounts list \
    --project=$GCP_PROJECT \
    --filter="displayName:$GCS_SA" \
    --format='value(email)')

gcloud projects add-iam-policy-binding $GCP_PROJECT \
    --role roles/storage.admin \
    --member serviceAccount:$GCS_SA_EMAIL

gcloud projects add-iam-policy-binding $PROJECT \
    --member serviceAccount:$GCS_SA_EMAIL \
    --role roles/browser
```

**Ukázka kódu 5.2:** Vytvoření účtu služby - gcs-service-account

Následně vytvoříme instanci virtuálního počítače s názvem Spinnaker-VM, ke kterému přiřadíme účet služby halyard-service-account, zvolíme geografickou oblast

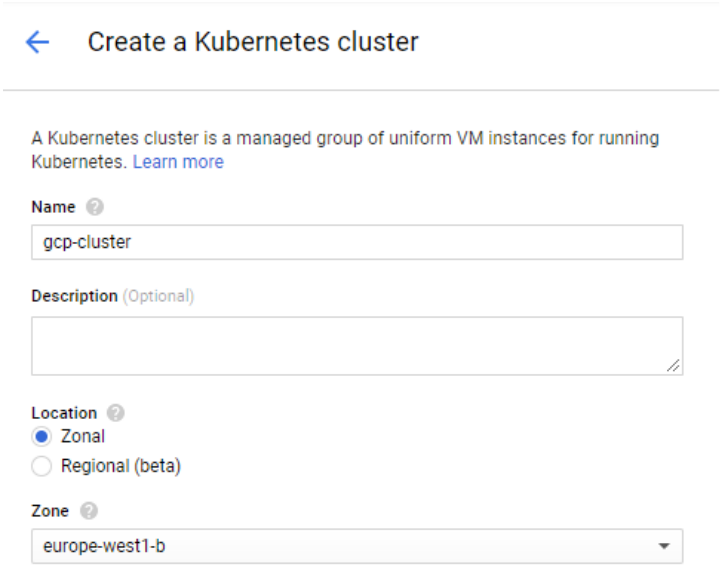
(us-central1-f), operační systém (Ubuntu 14.04 - doporučený pro Spinnaker) a výpočetní prostředky, jimiž bude disponovat. V tomto případě typ n1-standard-2, tedy dvoujádrový procesor s 7,5 GB paměti RAM. Virtuální počítač lze vytvořit grafickým rozhraním ve webovém prohlížeči nebo tímto příkazem v příkazovém řádku Google Cloud Shell:

```
gcloud compute instances create Spinnaker-VM \
    --project=$GCP_PROJECT \
    --zone=us-central1-f \
    --scopes=cloud-platform \
    --service-account=$HALYARD_SA_EMAIL \
    --image-project=ubuntu-os-cloud \
    --image-family=ubuntu-1404-lts \
    --machine-type=n1-standard-2
```

**Ukázka kódu 5.3:** Vytvoření virtuálního počítače v Google Cloud Platform

Virtuálnímu počítači přiřadíme veřejnou IP adresu a SSH klíč aby bylo možné se k němu připojit z lokálního počítače pomocí SSH protokolu.

Posledním krokem v přípravě prostředí v Google Cloud Platform je vytvoření Kubernetes clusteru. V grafickém prostředí využijeme záložku Kubernetes Engine, ve které stačí zadat název a vybrat geografickou oblast, kde bude cluster nasazen. Zvolíme odlišnou lokalitu než v případě virtuálního počítače, a to z toho důvodu, že v rámci předplatného zdarma lze využít pouze jednu veřejnou IP adresu v dané oblasti. Pro naši nenáročnou aplikaci není nezbytně nutné povolovat autoscaling, který dokáže horizontálně škálovat počet instancí, na kterých cluster běží, podle zátěže. Ostatní parametry tak můžeme ponechat ve výchozím stavu - vznikne jeden Master a tři Nody s dostačujícím výpočetním výkonem.




**Obrázek 5.2:** Vytvoření Kubernetes clusteru v Google Cloud Platform

### 5.1.2 Microsoft Azure

Pro vytvoření Kubernetes clusteru v prostředí Microsoft Azure je k dispozici šablona s názvem Azure Container Service v obchodě Azure Marketplace. Ale nejdříve je třeba vytvořit skupinu prostředků v určité geografické oblasti. Protože cluster v Google Cloud Platform je umístěn v europe-west1-b a my chceme nasadit geograficky distribuovanou aplikaci, zde vybereme servery v USA - konkrétně oblast s názvem West US.

Opět je možné rozhodnout, kolik Kubernetes Master a Node komponent bude vytvořeno - zvolíme stejný počet jako v předchozím případě, tedy jeden Master a tři Nody. Dále je nutné zadat DNS prefix clusteru, jméno administrátora a SSH klíč, kterým se ke clusteru připojíme. Poté se již cluster se všemi náležitostmi vytvoří.

| NAME | TYPE | LOCATION |
|---|---|---|
| 00xq7uvt4ev66qgagnt0 | Storage account | West US |
| agent-availabilitySet-F1D66E0F | Availability set | West US |
| azure | Container service | West US |
| k8s-agent-F1D66E0F-0 | Virtual machine | West US |
| k8s-agent-F1D66E0F-1 | Virtual machine | West US |
| k8s-agent-F1D66E0F-2 | Virtual machine | West US |
| k8s-agent-F1D66E0F-nic-0 | Network interface | West US |
| k8s-agent-F1D66E0F-nic-1 | Network interface | West US |
| k8s-agent-F1D66E0F-nic-2 | Network interface | West US |
| k8s-master-F1D66E0F-0 | Virtual machine | West US |
| k8s-master-F1D66E0F-nic-0 | Network interface | West US |
| k8s-master-F1D66E0F-nsg | Network security group | West US |
| k8s-master-F1D66E0F-routetable | Route table | West US |
| k8s-master-internal-lb-F1D66E0F | Load balancer | West US |
| k8s-master-ip-kubernetes-azuremgmt-F1D66E0F | Public IP address | West US |
| k8s-master-lb-F1D66E0F | Load balancer | West US |
| k8s-vnet-F1D66E0F | Virtual network | West US |
| kubernetes-azuremgmt | Load balancer | West US |

**Obrázek 5.3:** Kubernetes cluster v Microsoft Azure

## 5.2 Spinnaker - instalace

Po přípravě multi-cloud prostředí následuje instalace samotného Spinnakeru. Připojíme se k virtuálnímu počítači přes SSH tunnel, ve kterém je velmi důležité přesměrovat porty 9000 a 8084 ze vzdáleného virtuálního počítače na počítač lokální. Oba porty jsou nezbytné pro správnou funkčnost Spinnakeru, konkrétně port 9000 je pro připojení k webovému rozhraní Deck a port 8084 pro API bránu, která komunikuje s ostatními službami. Lokální přesměrování portů se provede přidáním parametru -L do příkazu pro připojení.

Výsledný příkaz je tedy:

```
ssh uzivatel@verejna_ip -L 9000:localhost:9000 -L 8084:localhost:
8084
```

Po úspěšném připojení se všechny následující instalace a nastavení vykonávají na vzdáleném virtuálním počítači.

### 5.2.1 Instalace nástrojů - Halyard, Docker a kubectl

V této podkapitole je popsána instalace nástroje Halyard, bez kterého by nebylo možné nasadit a spravovat Spinnaker a nástrojů Docker a Kubernetes kubectl potřebných pro nasazení webové aplikace do vybraného prostředí multi-cloudu.

#### Halyard

Halyard je nástroj zodpovědný za správu celého průběhu nasazení Spinnakeru. To zahrnuje přijímání změn konfigurace, nasazení jednotlivých dílčích komponent a provádění aktualizací nasazení. Halyard se nainstaluje v příkazovém řádku, kdy se nejdříve stáhne instalační skript a poté se spustí:

```
curl -O https://raw.githubusercontent.com/spinnaker/halyard/master/
install/debian/InstallHalyard.sh

sudo bash InstallHalyard.sh
```

**Ukázka kódu 5.4:** Instalace nástroje Halyard

Ověření instalace probíhá příkazem *hal -v*, po kterém se zobrazí právě nainstalovaná verze:

```
radek@spinnaker-vm:~$ hal -v
0.42.0-20180302093153
```

**Obrázek 5.4:** Halyard - ověření instalace

#### Docker

Druhým nástrojem, bez něhož by v další části této práce nebylo možné vytvořit Docker image, je Docker. K instalaci vybereme verzi Community Edition, která je vhodná pro samostatné vývojáře a menší týmy. Další možností je zpoplatněná verze Enterprise Edition, která je určená pro větší podniky a především pro ty, které provozují své kritické aplikace v Docker kontejnerech.

Docker podporuje všechny běžně používané operační systémy - Windows, macOS a linuxové distribuce Ubuntu, Debian, CentOS a Fedora.

Docker pro náš operační systém (Ubuntu) nainstalujeme pomocí návodu, který je dostupný na webové adrese: `https://docs.docker.com/install/linux/docker-ee/ubuntu/`

Následně instalaci ověříme tímto příkazem:

```
sudo docker run hello-world
```

Při vykonání příkazu Docker spustí kontejner obsahující image s názvem hello-world. Obrazy kontejnerů mohou být pojmenované různě, například podle verze, ale protože jsme žádný název neuvedli, použije se image s označením latest. Docker začne tím, že se ho pokusí najít lokálně ale neúspěšně, protože jde o jeho první sestavení, tudíž se stáhne z veřejného repozitáře Docker Hub. Po stažení se vytvoří kontejner a posléze se spustí skript uvnitř tohoto kontejneru, který vypíše uvítací zprávu a informace o úspěšné instalaci.

```
radek@spinnaker-vm:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:97ce6fa4b6cdc0790cda65fe7290b74cfebd9fa0c9b8c38e979330d547d22ce1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

**Obrázek 5.5:** Docker - Hello World (ověření instalace)

### Kubectl

Kubernetes clustery se konfigurují přes utilitu kubectl, která umožňuje nasazení a správu aplikací v Kubernetes. Dalším krokem je tedy instalace CLI rozhraní kubectl. Do proměnné KUBECTL_LATEST definujeme poslední dostupnou verzi, stáhneme instalační soubor, přidáme souboru práva na spouštění a přesuneme jej do adresáře */usr/local/bin/*, který je určen pro lokální software.

```
KUBECTL_LATEST=$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)
```

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/
$KUBECTL_LATEST/bin/linux/amd64/kubectl

chmod +x kubectl

sudo mv kubectl /usr/local/bin/kubectl
```

**Ukázka kódu 5.5:** Instalace nástroje kubectl

### 5.2.2 Úložiště - Google Cloud Storage

Spinnaker vyžaduje externí úložiště pro uchování perzistních dat z mikroslužby Front50 zahrnující nastavení nasazované aplikace a konfiguraci jednotlivých pipeline. Vzhledem k tomu, že tyto údaje bývají velmi citlivé a klíčové pro danou aplikaci, je vhodné použít zabezpečené cloudové úložiště - v našem případě Google Cloud Storage.

Nyní vygenerujeme JSON (JavaScript Object Notation) klíč k již existujícímu účtu služby, který je potřebný pro bezpečný přenos požadovaných dat do vybraného úložiště a zároveň bude sloužit i pro přístup do Google Container Registry. Nejprve vytvoříme adresář, kde bude klíč uložen, poté následuje přiřazení účtu služby a samotné vytvoření klíče s názvem *gcp.json*:

```
mkdir -p $(dirname ~/.gcp/gcp.json)

GCS_SA_EMAIL=$(gcloud iam service-accounts list \
    --filter="displayName:gcs-service-account" \
    --format='value(email)')

gcloud iam service-accounts keys create ~/.gcp/gcp.json \
    --iam-account $GCS_SA_EMAIL
```

**Ukázka kódu 5.6:** Vygenerování JSON klíče

Dále přidáme vybrané úložiště do konfigurace Halyard, kde jsou vyžadovány parametry s názvem projektu na Google Cloud Platform a cesta k vygenerovanému JSON klíči:

```
hal config storage gcs edit \
    --project $(gcloud info --format='value(config.project)') \
    --json-path ~/.gcp/gcp.json
```

**Ukázka kódu 5.7:** Halyard - přiřazení úložiště Google Cloud Storage

A nakonec změníme typ úložiště na Google Cloud Storage:

```
hal config storage edit --type gcs
```

### 5.2.3 Nastavení poskytovatelů

Spinnaker provádí všechny funkce průběžného doručování aplikací prostřednictvím providerů. Aby Spinnaker tedy mohl cokoliv udělat, musí být přidán alespoň jeden poskytovatel s aktivním uživatelským účtem.

#### Docker Registry

Docker Registry zde není brán doslova jako provider, ale funguje pouze jako zdroj (repozitář) pro Docker image, tudíž neumožňuje nasazení image. Spinnaker podporuje repotizáře: DockerHub, Google Container Registry, Amazon Elastic Container Registry a další.

V Halyard povolíme poskytovatele docker-registry a přidáme účet s názvem my-gcr-account. Adresa pro Google Container Registry je gcr.io a znovu využijeme JSON klíč pojmenovaný *gcp.json*.

```
hal config provider docker-registry enable

hal config provider docker-registry account add my-gcr-account \
    --address gcr.io \
    --password-file ~/.gcp/gcp.json \
    --username _json_key
```

**Ukázka kódu 5.8:** Halyard - Docker Registry

#### Kubernetes - Microsoft Azure

Pro připojení ke Kubernetes clusteru na Microsoft Azure je nutné nainstalovat rozhraní příkazového řádku s názvem Azure CLI, které nabízí prostředí pro správu prostředků v portálu Azure. Instalace Azure CLI:

```
echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-
cli/lsb_release -cs main" | sudo tee /etc/apt/sources.list.d/
azure-cli.list

sudo apt-key adv --keyserver packages.microsoft.com --recv-keys
52E16F86FEE04B979B07E28DB02C46DF417A0893

sudo apt-get install apt-transport-https

sudo apt-get update && sudo apt-get install azure-cli
```

**Ukázka kódu 5.9:** Instalace Azure CLI

Po úspěšné instalaci se přihlásíme do Microsoft Azure pomocí *az login*. Připojení ke Kubernetes clusteru se provede následujícím příkazem, kde zadáme název skupiny prostředků a jméno clusteru:

```
az acs kubernetes get-credentials --resource-group=kubernetes-
cluster --name=azure
```

Díky tomu můžeme přímo z lokálně ovládaného virtuálního počítače přistupovat do nového Kubernetes clusteru. Stáhne se konfigurace clusteru a prostřednictvím kubectl je možné ověřit funkčnost připojení a případně zobrazit počet běžících Nodů nebo služeb.

V Halyard povolíme Kubernetes jako poskytovatale a přidáme účet s názvem *my-k8s-azure* z důvodu rozlišení, že se jedná o cluster na Microsoft Azure. Do parametrů přidáme účet pro Docker Registry a aktuální konfiguraci kubectl.

```
hal config provider kubernetes enable

hal config provider kubernetes account add my-k8s-azure \
    --docker-registries my-gcr-account \
    --context $(kubectl config current-context)
```

**Ukázka kódu 5.10:** Halyard - Kubernetes cluster v Microsoft azure

### Kubernetes - Google Cloud Platform

Připojení ke Kubernetes clusteru v GCP probíhá podobně jako v předchozím případě, ale protože se nachází na stejné platformě jako virtuální počítač, není třeba instalovat další podpůrné nástroje. Stačí nastavit použití klientského certifikátu a získat pověření k připojení ke clusteru, přičemž musíme zadat jeho jméno a zónu umístění.

```
gcloud config set container/use_client_certificate true

gcloud container clusters get-credentials gcp-cluster \
    --zone=europe-west1-b
```

**Ukázka kódu 5.11:** Připojení ke Kubernetes clusteru v Google Cloud Platform

Znovu je možné spojení otestovat utilitou kubectl a získat tak informace o nově připojeném clusteru. Poskytovatel Kubernetes je již povolený, pouze k němu přidáme nový účet s názvem *my-k8s-google* a stejně jako u Microsoft Azure účet pro Docker Registry a aktuální konfiguraci kubectl.

```
hal config provider kubernetes account add my-k8s-google \
    --docker-registries my-gcr-account \
```

```
--context $(kubectl config current-context)
```

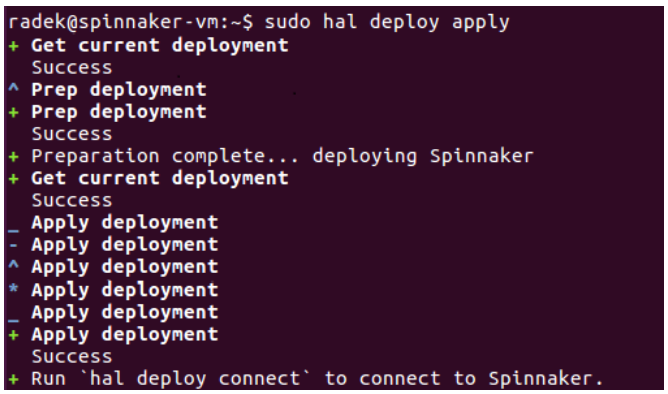
**Ukázka kódu 5.12:** Halyard - Kubernetes cluster v Google Cloud Platform

### 5.2.4 Nasazení

Po dokončení nastavení úložiště a poskytovatelů zbývá už jen zvolit verzi (v našem případě je to nejnovější verze - 1.6.0), kterou chceme nasadit a poté příkazem hal deploy apply dojde k nasazení všech mikroslužeb Spinnakeru.

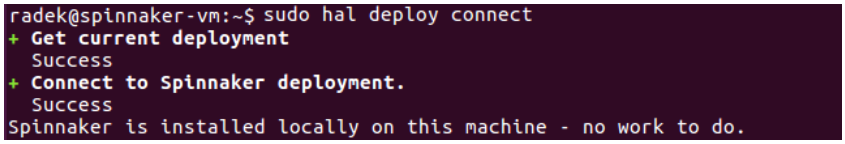
```
hal config version edit --version $(hal version latest -q)
```

**Ukázka kódu 5.13:** Halyard - nasazení poslední dostupné verze



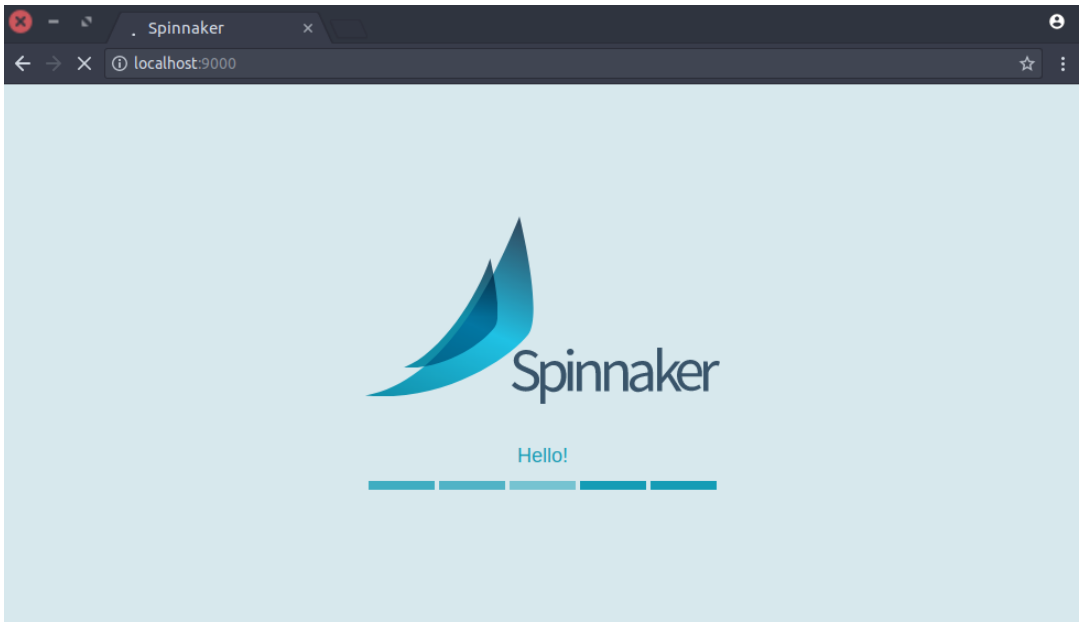

**Obrázek 5.6:** Spinnaker - nasazení

Nakonec se k nasazení připojíme, což proběhne velmi rychle, protože je Spinnaker nainstalován lokálně na virtuálním počítači a není tak potřeba se připojovat ke vzdáleným mikroslužbám.




**Obrázek 5.7:** Spinnaker - připojení k nasazení

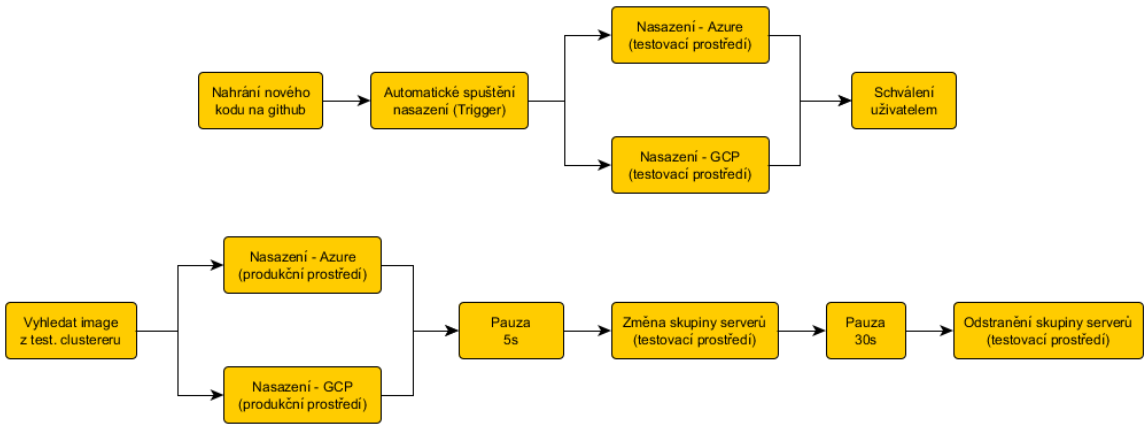
Grafické uživatelské rozhraní zaručující vždy stejný způsob spouštění automatizovaných procesů a současně usnadňuje práci nejen vývojovému týmu, ale umožní spustit proces nasazení aplikace i ostatním členům (např. management), kteří příkazový řádek neznají a není pro ně vhodným způsobem ovládání, je dostupné přes webový prohlížeč na adrese localhost s portem 9000.




**Obrázek 5.8:** Spinnaker - uživatelské rozhraní

## 5.3 Implementace aplikace do prostředí multi-cloud

Implementace bude probíhat ve dvou etapách, které v podstatě následně okopírují vytvořené pipeline.




**Obrázek 5.9:** Proces implementace aplikace do prostředí multi-cloud

První fáze zahrnuje počáteční změnu kódu a jeho nahrání na GitHub, nasazení aplikace do testovacího prostředí a končí schválením uživatele, zda se dané řešení z testovacího prostředí nasadí i do produkčního prostředí.

Druhá fáze zahrnuje vyhledání image z clusteru v testovacím prostředí, jeho nasazení do produkčního prostředí a poté odstranění skupiny serverů v testovacím prostředí, protože

již nebudou potřeba. Je zde zahrnuta i fáze změny skupiny serverů, která pouze potlačí jejich funkčnost, ale nedojde k odstranění. U změny a odstranění skupiny serverů jsou vloženy dvě časové prodlevy, a to z důvodu vyzkoušení více funkcí Spinnakeru nebo případně, aby mohl uživatel odstranění přerušit.

### 5.3.1 Vytvoření Docker image

V této části je nakonfigurován Container Builder, který rozpozná změny ve zdrojovém kódu aplikace, vytvoří Docker image a nahraje do Google Container Registry.

Začneme naklonováním již dříve definovaného repozitáře z GitHub a následným přesunem do nově vzniklé složky. Příkazem git checkout založíme novou větev s označením release, do které se rovnou přepneme přidáním přepínače -b a zároveň zjistíme, že všechny soubory z repozitáře jsou aktuální:

```
git clone https://github.com/Rada236/nginx-multicloud.git

cd nginx-multicloud

git checkout -b release
```

**Ukázka kódu 5.14:** Klonování repozitáře z GitHub

Protože bude nasazení aplikace probíhat z Docker image, musíme jej nejdříve vytvořit a spolu s ním i spouštěcí Trigger, pomocí kterého se při změně kódu automaticky zahájí nasazení.

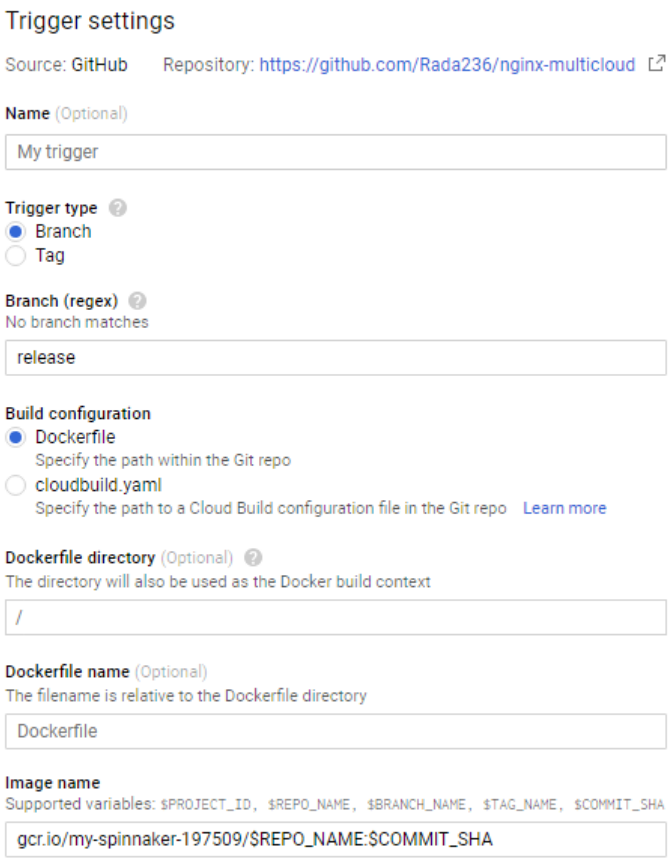
V adresáři nginx-multicloud vytvoříme z Dockerfile Docker image s názvem nginx:latest a nahrajeme jej do Google Container Registry:

```
sudo docker build -t gcr.io/my-spinnaker-197509/nginx:latest .

sudo gcloud docker -- push
   "gcr.io/my-spinnaker-197509/nginx:latest"
```

**Ukázka kódu 5.15:** Vytvoření Docker image

Trigger vytvoříme ve webovém rozhraní v Google Cloud Platform přes záložku Container Registry - Build Triggers. Vybereme zvolené úložiště kódu, v tomto případě GitHub, a poté zadáme název Triggeru, tvorbu přes Dockerfile a pojmenování větvení v GitHub.



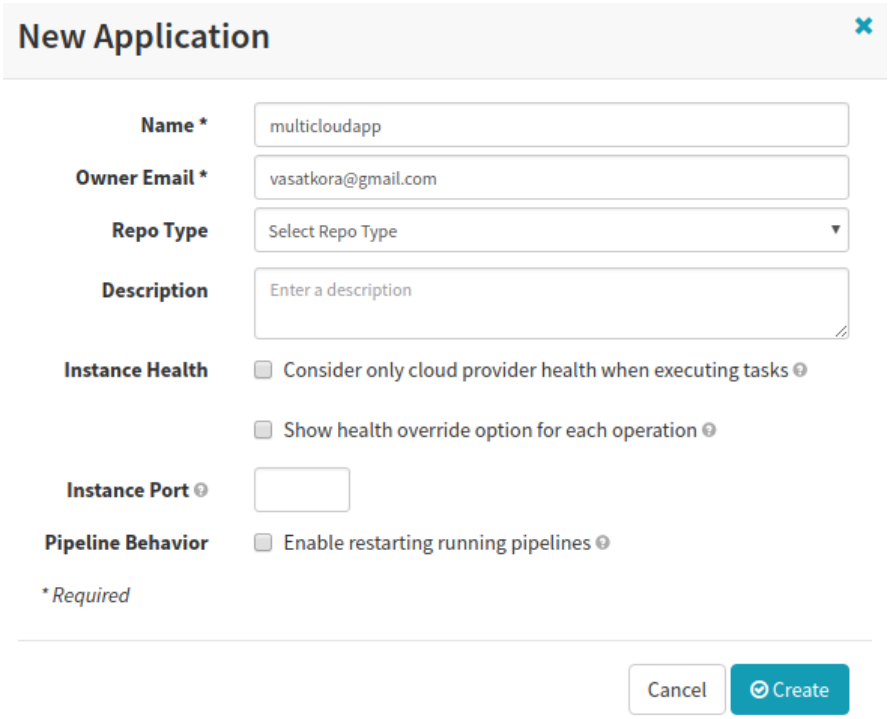

**Obrázek 5.10:** Trigger v Google Cloud Platform

Od tohoto okamžiku, kdykoli dojde k odeslání změněného zdrojového kódu do větve release v GitHub, nástroj Container Builder automaticky vytvoří a nahraje kód aplikace jako Docker image do Google Container Registry.

### 5.3.2 Vytvoření aplikace

Nyní, když jsou splněny všechny náležitosti automatického vytváření Docker image, následuje konfigurace nasazované aplikace. Aplikace je vždy vázaná na Git repozitář (Github nebo Bitbucket), tudíž pokaždé odkazuje na stejný zdrojový kód.

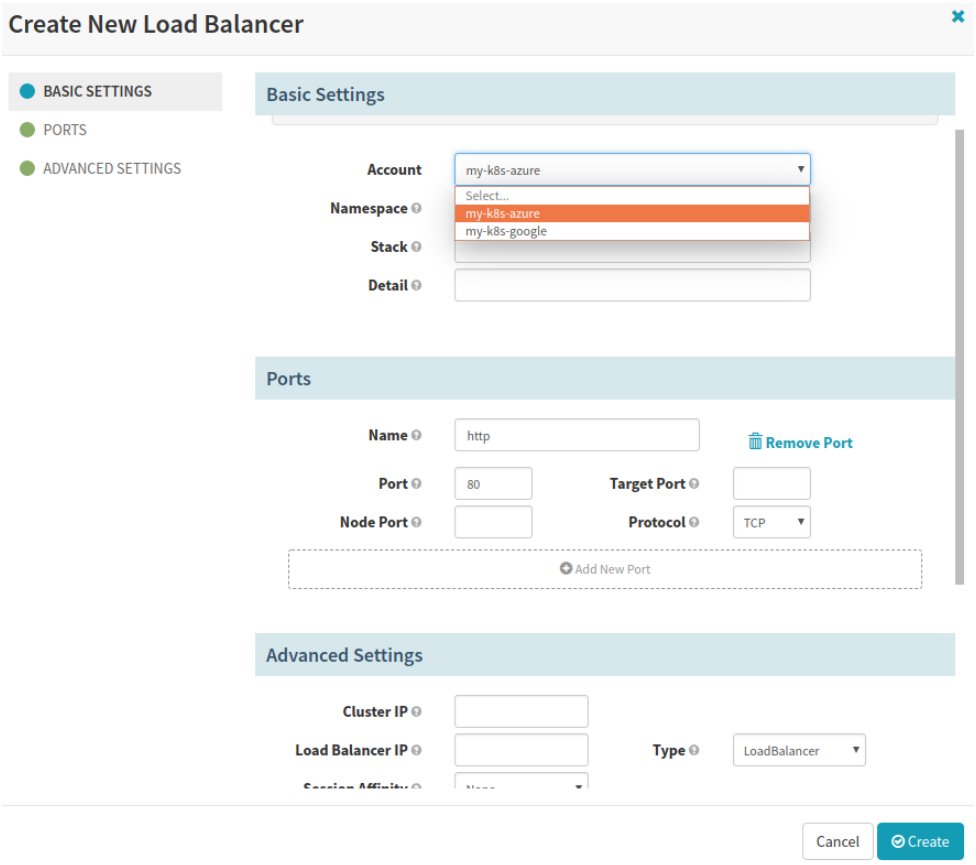
Přesuneme se do webového rozhraní Spinnakeru, kde vytvoříme aplikaci s názvem multicloudapp a vyplníme e-mailovou adresu vlastníka aplikace. Zbylé nepovinné informace není nutné vyplňovat.




**Obrázek 5.11:** Spinnaker - vytvoření aplikace

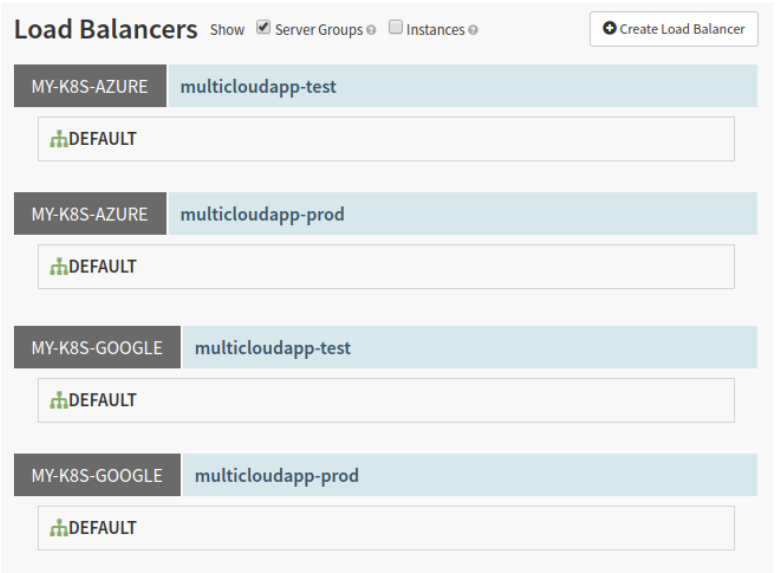
### 5.3.3 Load Balancery

V sekci Load Balancers připravíme dva load balancery s účtem my-k8s-google na GCP a dva s my-k8s-azure na Microsoft Azure. U obou poskytovatelů je jeden určen pro testovací prostředí a druhý pro produkční prostředí. Ve webovém rozhraní vybereme požadovaný účet, napíšeme název load balenceru, zadáme parametry portu - HTTP, číslo portu 80 a protokol TCP. Nakonec změníme typ na LoadBalancer a zbývající parametry ponecháme nevyplněné nebo ve výchozím stavu.




**Obrázek 5.12:** Spinnaker - vytvoření load balanceru

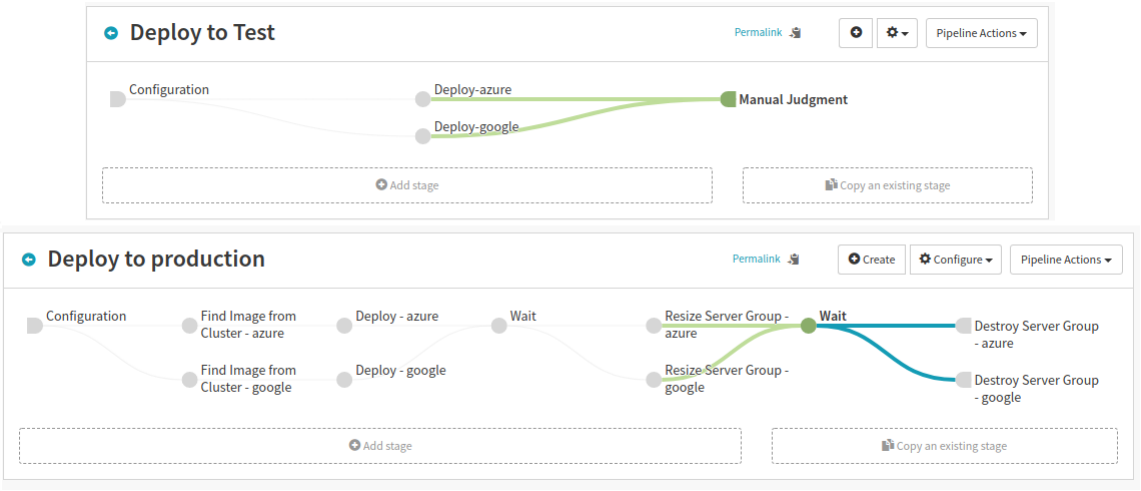
Celkem vzniknou 4 load balancery (z pohledu Kubernetes se jedná o služby, ve kterých poběží nasazená aplikace) - označené dle názvu aplikace a prostředí, v němž se nachází, tudíž dva pojmenované *multicloudapp-test* a dva *multicloudapp-prod*.




**Obrázek 5.13:** Spinnaker - nově vytvořené load balancery

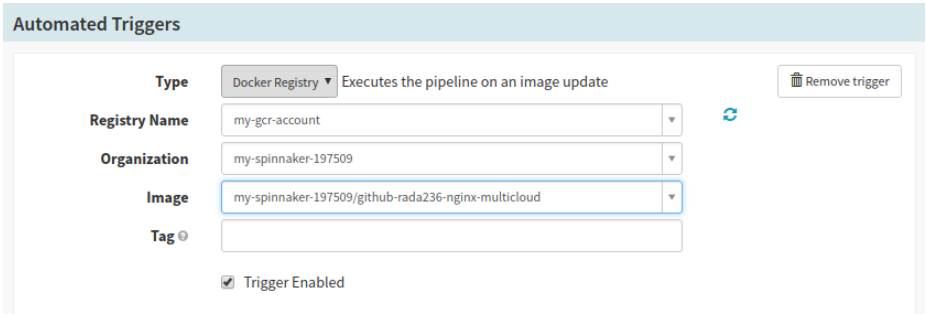
### 5.3.4 Pipeline

Jak již bylo zmíněno, vytvoříme dvě pipeline - první Deploy to Test je určena pro testovací prostředí a druhá Deploy to Production nasadí aplikaci do prostředí produkčního.



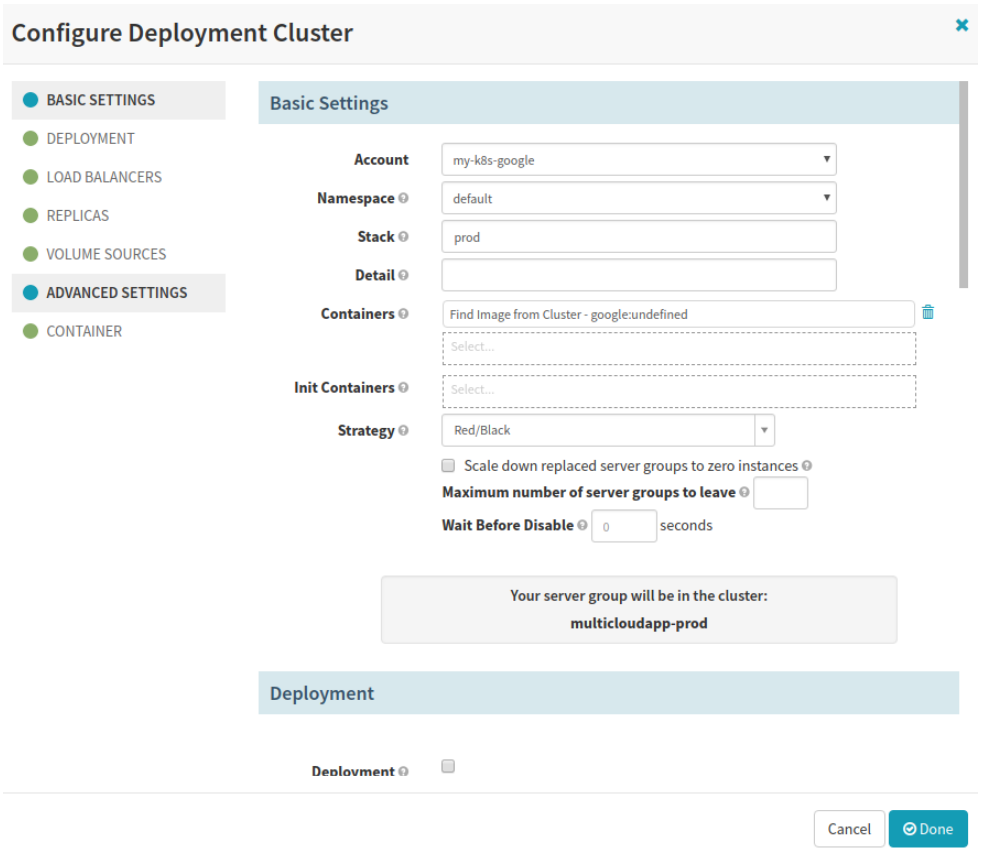

**Obrázek 5.14:** Spinnaker - pipeline

V první pipeline je nastaveno automatické spuštění nasazení, které začne, jestliže se objeví nová verze image v Docker Registry v GCP. Změna image je propojena s účtem na GitHub přes dříve vytvořený Trigger, to znamená, že v okamžiku, kdy dojde k nahrání nového kódu na GitHub, změní se i image v Docker Registry. Dále pipeline obsahuje nasazení aplikace prostřednictvím strategie Red/Black do testovacího prostředí v Google Cloud Platform a Microsoft Azure a následnou validaci od uživatele má-li se v nasazení pokračovat. Potvrzení od uživatele je vyžadováno kvůli kontrole funkčnosti nasazované aplikace. Fáze ověření by se dala nahradit pauzou, ve které by kontrola proběhla, a nasazení pokračovalo bez zásahu uživatele. V této pauze by se, při případné chybě, mohl proces nasazení aplikace také zastavit.




**Obrázek 5.15:** Spinnaker - trigger

Druhá pipeline se automaticky spustí po úspěšném dokončení první pipeline, tedy po manuálním schválení uživatelem. Dále následuje vyhledání image z testovacího prostředí a nasazení těchto image do produkčního prostředí. K nasazení v Google Cloud Platform i v Microsoft Azure je využita strategie Red/Black z důvodu možného obnovení předchozích verzí aplikace. Poté dojde ke změně skupiny serverů na přesný počet - nastaveno na nula instancí, to znamená jejich vypnutí. A nakonec se odstraní oba clustery určené pro testovací prostředí.




**Obrázek 5.16:** Spinnaker - nasazení do produkčního prostředí

Změna skupiny serverů se využívá především při škálování aplikace, kdy je možné aplikaci takzvaně utlumit na požadovaný počet instancí. V našem případě je tato fáze pouze pro ukázku, že Spinnaker funkci nabízí, protože webová aplikace využívá pouze jednu instanci.

## 5.4 Nasazení aplikace

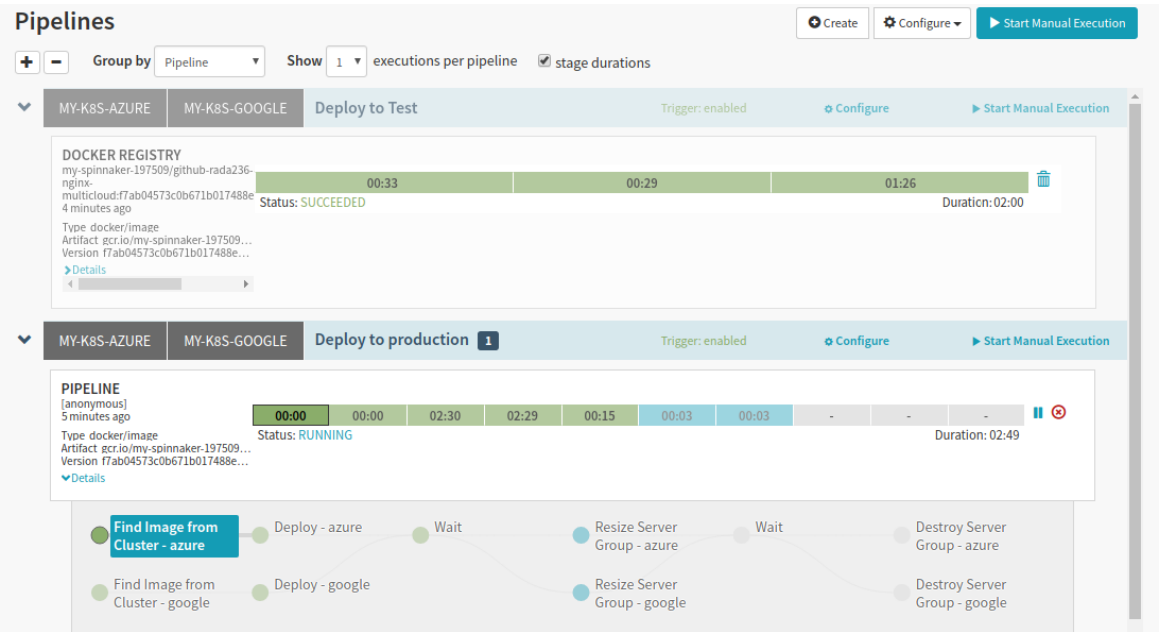
Po vytvoření aplikace, load balancerů a pipeline - je vše připraveno pro průběžné nasazení softwaru.

Sekvence událostí je následující:

1. Nahrání nového nebo změněného kódu na GitHub do větve release.
2. Automatické vytvoření Docker image v Google Container Registry.
3. Spinnaker automaticky nasadí aplikaci do testovacího prostředí na Kubernetes cluster v Google Cloud Platform a Microsoft Azure.
4. Uživatel zkontroluje nasazení aplikace v testovacím prostředí a potvrdí nasazení do produkčního prostředí.
5. Spinnaker automaticky nasadí aplikaci z testovací fáze do produkčního prostředí na Kubernetes cluster v GCP a Microsoft Azure.
6. Spinnaker automaticky odstraní oba clustery z testovacího prostředí.

### 5.4.1 Prvotní manuální nasazení

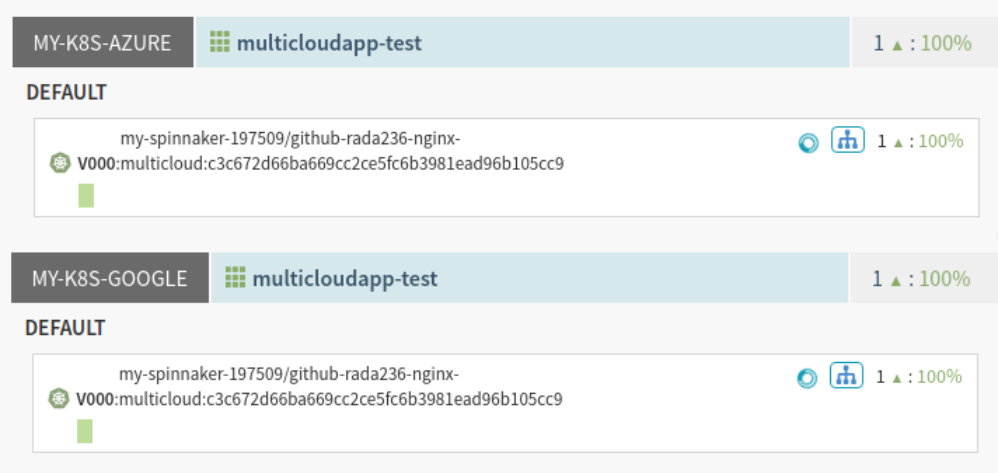
Proces nasazení je kromě automatického spuštění možné zahájit i manuálně. Manuální start je vhodný především při prvotním nasazení aplikace, tedy v době kdy, je připraven Docker image s aktuálním zdrojovým kódem a není nutné nic měnit. Ve Spinnakeru je volba manuálního zahájení obsažena ve webovém uživatelském rozhraní, kde u první pipeline Deploy to Test, stačí zvolit Start Manual Execution a nasazení započne.




**Obrázek 5.17:** Spinnaker - průběh nasazení

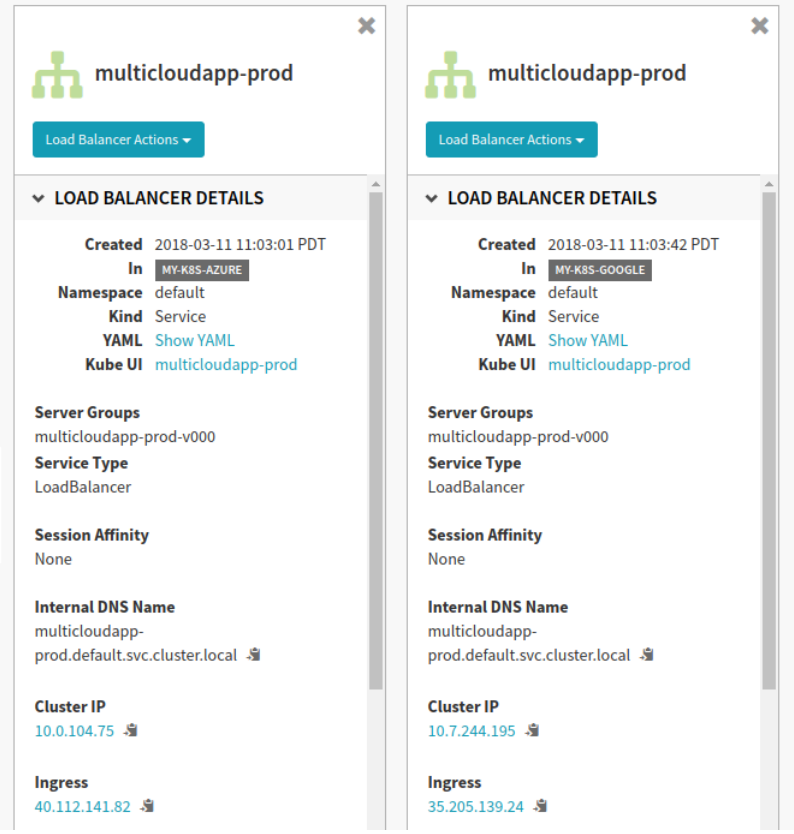
U jednotlivých fází v pipeline můžeme vidět délku jejich trvání a grafické rozlišení, jestli jsou právě vykonávané (modře), úspěšně dokončené (zeleně), nedokončené (červeně) nebo čekají na potvrzení od uživatele (oranžově). Dále je možné zobrazit detailní informace u všech fází nasazení a také celý průběh nasazení pozastavit nebo ukončit.

V průběhu nasazení aplikace se automaticky vytvoří clustery - dva pro testovací prostředí po dokončení první pipeline a dva pro produkční prostředí v průběhu druhé pipeline. Clustery určené pro testování aplikace budou po nasazení aplikace do produkčního prostředí odstraněny.



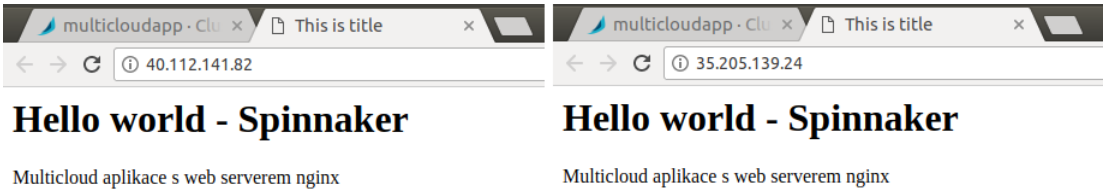

**Obrázek 5.18:** Spinnaker - automaticky vytvořené clustery v testovacím prostředí

U clusterů je možné zobrazit detailní informace, jako například čas a datum vytvoření, k jakému účtu patří, výpis konfigurace v jazyce YAML, do jaké skupiny serverů náleží, interní IP adresu a DNS, veřejnou IP adresu a další. Veřejná IP adresa se skrývá pod označením "Ingress". Ingress v Kubernetes nabízí způsob, jak centralizovat řadu služeb do jediného vstupního bodu neboli dané službě přiřadí veřejnou IP adresu a není tak nutné odkazovat na celý cluster. To znamená, že na jednom clusteru může být více služeb s rozdílnou veřejnou IP adresou.




**Obrázek 5.19:** Spinnaker - detailní informace clusterů

Nasazená aplikace, v tomto případě statická HTML stránka Hello World, je dostupná přes veřejné IP adresy Kubernetes služeb na příslušných clusterech. Stačí tedy zadat IP adresy (Ingress) do webového prohlížeče (pro GCP 40.112.141.82 a pro Microsoft Azure 35.205.139.24) a zobrazí se námi nasazená aplikace.




**Obrázek 5.20:** Prvotní nasazení aplikace - Hello World

### 5.4.2 Automatické nasazení

V této části je uskutečněno automatické nasazení aplikace do prostředí multi-cloudu spuštěného při každém odeslání změněného kódu do repozitáře Git. Nahrání kódu rozpozná Trigger a nástroj Container Builder vytvoří nový Docker image, ze kterého bude aplikace znovu nasazena.

Automatické nasazení tedy vždy začíná změnou kódu aplikace. Po úpravě zdrojového kódu zkontrolujeme příkazem git checkout release, zda jsou v pracovním adresáři nezapsané změny, které kolidují s větví release, jejíž přezkoumání se provádí. Následně vybereme upravené soubory, které mají být odeslány, a přidáme zprávu, podle níž bude případně možné rozpoznat, jaká změna byla provedena.

```
git add Dockerfile html/index.html

git commit -m 'Prvni zmena kodu'
```

**Ukázka kódu 5.16:** Odeslání změněného kódu na GitHub

Protože byl repozitář vytvořen klonováním, odesílání probíhá na server origin, což je centrální bod a výchozí název, který Git dává serveru, z nějž je repozitář naklonován. Změněný kód tedy nahrajeme na server origin do větve release tímto příkazem (při vykonání příkazu je ještě nutné zadat uživatelské jméno a heslo k účtu na GitHub):

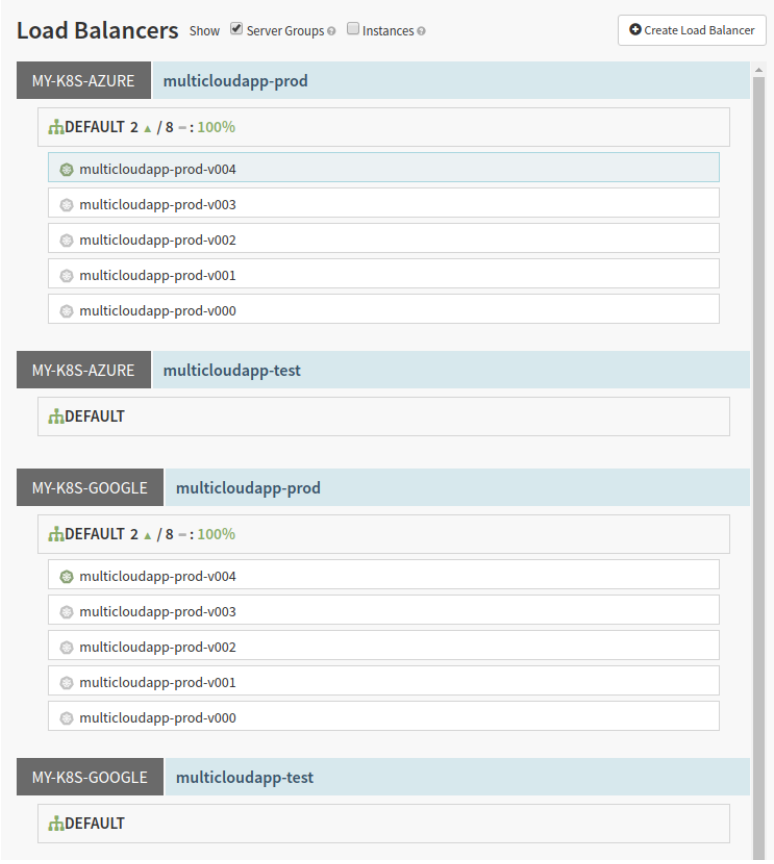
```
git push origin release
```

Po úspěšném nahrání nového kódu Spinnaker identifikuje pomocí Triggeru nový Docker image a automaticky se spustí první pipeline Deploy to Test. Druhá pipeline Deploy to Production začne také automaticky hned po úspěšném dokončení první pipeline, tedy po nasazení aplikace do testovacího prostředí a manuálním potvrzením od uživatele.

V průběhu každého nasazení se opět vytvoří clustery strategií nasazení Red/Black, to znamená, že se přidá do load balanceru nový cluster se změněnou verzí aplikace a předchozí se zablokuje, ale neodstraní.

Po kontrole aplikace ve webovém prohlížeči jsme zjistili, že je automatické spuštění a celý proces nasazení funkční a už se tak můžeme starat pouze o psaní kódu a vývoj samotné aplikace. Při nasazení nás pouze zajímá ověření funkčnosti v testovacím prostředí a manuální potvrzení, že se může v nasazení pokračovat.

Dohromady jsou odeslány do GitHub čtyři větší změny kódu aplikace, tudíž u obou load balancerů v produkčním prostředí můžeme vidět celkem pět verzí naší aplikace (je zde zahrnuto i prvním manuální spuštění). Load balancery v testovacím prostředí žádné zdroje neobsahují, protože jejich clustery jsou na konci druhé pipeline odstraněny.




**Obrázek 5.21:** Spinnaker - load balancery

### Finální nasazení aplikace

Po konečné změně obsahuje HTML kód webové aplikace dvě dynamické funkce, a to zobrazení aktuálního data a času a informace o IP adrese webového serveru, která se vypíše po kliknutí na příslušné tlačítko. Obě funkce jsou napsané v Javascriptu. Javascript je programovací jazyk, který dokáže udělat webovou stránku interaktivní. Klient, většinou internetový prohlížeč, si stáhne Javascriptový kód ze serveru jen jednou a se serverem již komunikovat nepotřebuje, protože tento kód je dále obsluhovaný klientským prohlížečem.

Finální HTML kód v souboru index.html:

```
<html>
 <header><title>Spinnaker</title></header>
<body>
  <div align="center">
    <h1> Spinnaker - Multicloud </h1>
    <img src="https://seeklogo.com/images/S/spinnaker-logo-
    32C3D1CDDF-seeklogo.com.png" alt="Spinnaker Logo" >

    <p>Aplikace nasazena do prostredi - Microsoft Azure a Google
```

```
    Cloud platform.</p>

    <b>Cas a datum:</b>
    <p id="date"></p>
      <script>
       document.getElementById("date").innerHTML = Date();
      </script>

    <p><b>Web server Nginx:</b></p>
    <button onclick="myFunction()">IP adresa</button>
    <p id="ip"></p>
      <script>
        function myFunction() {
        var x = location.hostname;
        document.getElementById("ip").innerHTML= x;
        }
      </script>
</div>
</body>
</html>
```

**Ukázka kódu 5.17:** Finální HTML kód webové aplikace

Výsledná aplikace, zahrnující všechny náležitosti definované v předchozí kapitole, je opět dostupná přes veřejné IP adresy na příslušných clusterech.

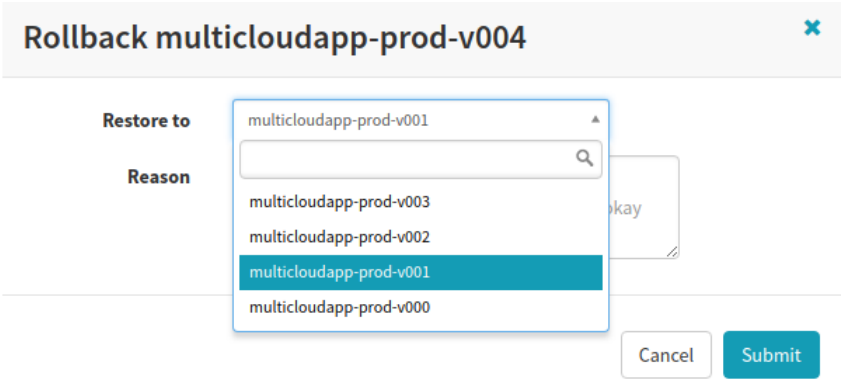

**Obrázek 5.22:** Výsledná aplikace nasazená na Kubernetes cluster v Google Cloud Platform

## 5.5 Návrat k předchozí verzi

Jestliže nastane nějaký problém s aktuálně nasazenou aplikací - Spinnaker nabízí možnost rychlého návratu (Rollback) k jakékoli předchozí verzi, ale pouze v případě, že nebyla smazána odpovídající skupina serverů, kam byla daná verze nasazena. Není tedy třeba vracet se postupně po jednotlivých verzích, ale lze v podstatě okamžitě obnovit aplikaci i do počátečního stavu. Rychlost obnovení záleží na složitosti aplikace.

Jak již bylo uvedeno, aby byl návrat k předchozí verzi možný, musí být zvolena strategie nasazení, která tuto funkci umožňuje, v tomto případě strategie Red/Black.

Návrat k předchozí verzi je dostupný ve webovém rozhraní Spinnakeru v sekci Load Balancers po označení libovolné skupiny serverů a otevření nabídky Server Group Actions. Následně stačí vybrat, do jakého stavu chceme aplikaci obnovit a akci potvrdit. Protože je aplikace nasazená u dvou poskytovatelů, je důležité ji vrátit do stejného stavu v Google Cloud Platform i v Microsoft Azure, jinak by nastala situace, že na každém z clusterů běží jiná verze aplikace.




**Obrázek 5.23:** Spinnaker - Rollback

# 6 Vyhodnocení nasazení aplikace

Z provedené implementace vyplývá, že Spinnaker je velmi užitečný nástroj pro dodání aplikace do prostředí multi-cloudu. Především významně usnadňuje koordinaci procesů zavádění a nasazování u více cloudových providerů s podobným rozhraním a navíc pomáhá normalizovat způsob vytváření jednotlivých pipeline, které jsou přehledné a transparentní v tom, jak se kód pohybuje napříč prostředím.

Spinnaker nabízí šablony pro rychlé nasazení v GCP, AWS i Microsoft Azure, ale toto řešení je omezené a vhodné pouze pro nasazení aplikací u poskytovatele, na kterém Spinnaker běží - nevyhovuje multi-cloudu.

Pro implementaci do multi-cloudu jsme proto museli uplatnit instalaci skrz Halyard, který nabízí rozšířené možnosti nastavení. Největší komplikace nastaly právě při instalaci a nastavení Spinnakeru, kdy nejdříve bylo nutné vyřešit vzájemné propojení s úložištěm perzistentních dat, s repozitářem Google Container Registry a hlavně s Kubernetes clustery. I s využitím oficiální dokumentace Spinnakeru je mnoho příležitostí, kde udělat chybu. Také pochopení samotných dokumentů není snadné.

Spinnaker samozřejmě nabízí spoustu dalších možností a funkcí, které lze přidat a nakonfigurovat, ale toto námi vykonané řešení mělo ukázat základní myšlenku, jak začít a úspěšně nasadit aplikaci do prostředí multi-cloudu. Spinnaker se stále vyvíjí a roste, aby podpořil co největší uživatelskou základnu než pouze Netflix, kvůli kterému vznikl, a proto často vychází různé aktualizace přinášející nové funkce.

Stejně jako u všech nových nástrojů je třeba si ze začátku zvyknout na způsob užívání a získávání informací ve webovém uživatelském rozhraní. Ale jakmile se pochopí smysl, jak Spinnaker v nasazení postupuje, je nastavení jednotlivých komponent a provedení případných změn poměrně snadné a časově nenáročné. Významným způsobem k tomu pomáhá funkce kopírování pipeline, která podstatně zkrátí dobu konfigurace dalších podobných pipeline. Složitost se zvyšuje především tím, na kolik cloudových poskytovatelů chceme aplikaci nasadit.

Co je ovšem trochu matoucí - Spinnaker ve výchozím nastavení nepoužívá pro nasazení stejné objekty jako jsou v Kubernetes. Z čehož vyplývá, že namísto využití principů Kubernetes nasazení Spinnaker použije k provedení pokročilých strategií nasazení vlastní logiku. To může způsobit problémy při přesunu stávajícího, již užívaného Kubernetes nastavení na Spinnaker.

Další, s tím související nejasností, jež mohla vést k určitým nesrovnalostem, je pojmenování komponent ve Spinnakeru, které je odlišné než v případě Kubernetes. Pravděpodobnou příčinou je, že zpočátku byl Spinnaker určen pouze pro nasazení skrz virtuální počítače (instance) a až později byl stejný koncept použit i pro nasazení kontejnerů v Kubernetes. Projevuje se to například v případě Kubernetes služeb, které jsou ve Spinnakeru označeny jako load balancery. Dále je nasazení aplikace ve Spinnakeru pojmenováno jako instalace, Docker image jako služba a instance ve Spinnakeru odpovídá podu v Kubernetes. Našly by se i další velmi odlišné názvy.

Problémem může být i nemožnost uložení a zálohování pipeline nebo její načtení v jiném projektu, případně aplikaci. Lze pouze porovnat historie verzí. Konfigurace pipeline je uložena v mikroslužbě Front50, ale Spinnaker nenabízí možnost, jak ji uložit nebo načíst. Jediným východiskem, jak získat konfiguraci pipeline, je při zobrazení JSON souboru - ten je dostupný ve webovém rozhraní v nabídce nastavení u dané pipeline. Ale toto řešení je velmi těžkopádné, kód se musí vykopírovat, vložit, nahrát a nakonec znovu spustit celý proces nasazení Spinnakeru. Nehledě na to, že jakákoliv chyba či překlep při kopírování způsobí nefunkčnost celé pipeline.

Z toho vyplývá, že pipeline ve Spinnakeru nejsou neměnné - lze je vytvářet, spravovat a konfigurovat, ale nelze jejich nastavení uložit a načíst. Velkou nevýhodou je to zejména v případě, že nasazení selže a je tak nutné začít s konfigurací znovu od začátku. Do budoucna se počítá s využitím určitých konfigurovatelných šablon, které by tento problém měly vyřešit.

Poslední výtka, která byla objevena v průběhu jednotlivých nasazení, se týká pauz. Do druhé pipeline byly vloženy dvě časové prodlevy s délkou 5 a 30 sekund. U delší bylo vše v pořádku a proběhla přesně, jak měla. Avšak kratší pauza nikdy netrvala 5, ale 15 sekund. Nemá tedy smysl vkládat menší pauzy než zmiňovaných 15 s, protože nebudou přesně dodrženy.

Kromě těchto menších problémů a výhrad je Spinnaker komplexním a efektivním nástrojem pro snadné nasazení aplikací do prostředí multi-cloudu s využitím Docker kontejnerů. Nicméně v případě implementace softwaru pouze na jednoho veřejného poskyto-

vatele cloudu je vhodné zvolit jeho nabízené a již připravené řešení, které není nutné složitě konfigurovat.

## 6.1 Zhodnocení metrik nasazení

Podle dříve popsaných etap implementace softwaru se námi zvolené řešení nachází na rozhraní mezi průběžným doručováním a průběžným nasazením. To znamená, že nasazení probíhá průběžně jako proces v pozadí, dochází pouze k minimálním výpadkům aplikace a implementace je velmi rychlá. Ale zároveň je vyžadován manuální zásah uživatele. Bez této uživatelské akce by se již přímo jednalo o fázi průběžného nasazení.

### Čas do nasazení v produkčním prostředí

Čas od nahrání kódu do nasazení v produkčním prostředí je závislý zejména na tom, jak dlouho trvá kontrola aplikace v testovacím prostředí a následné uživatelské potvrzení. U všech provedených nasazení byla celková doba automaticky prováděných fází přibližně stejná, a to kolem 6 minut. Nasazení aplikace strategií Red/Black do testovacího i produkčního prostředí probíhalo zhruba jednu minutu. Nejdelší částí bylo odstranění testovací skupiny serverů, a to i z důvodu dvou vložených pauz.

### Počet uživatelských zásahů

Od odeslání změněného zdrojového kódu aplikace je nutné v průběhu procesu nasazení vykonat pouze jeden manuální zásah - jak již bylo několikrát zmíněno, jde o potvrzení, zda v nasazení pokračovat a je vyžadováno kvůli kontrole funkčnosti aplikace. Toto ověření je provedeno pomocí prostého kliknutí na příslušné tlačítko ve webovém rozhraní Spinnakeru. Všechny ostatní fáze probíhají automaticky.

### Návrat k předchozí verzi

Spinnaker nabízí snadné obnovení do jakékoli předchozí verze aplikace. Ale je nutné zvolit strategii nasazení, která tuto funkci podporuje. Je to tedy jedna z velkých výhod, protože podobné nástroje většinou návrat do předchozí verze ani neobsahují.

### Počet použitých nástrojů

Pro úspěšné průběžné dodávání softwaru do prostředí multi-cloudu byly použity nástroje GitHub, Halyard, Docker, Kubernetes a Spinnaker. Celkem tedy bylo využito 5 nástrojů v procesu nasazení aplikace.

# 7 Závěr

Cílem této práce bylo analyzovat a navrhnout multi-cloud platformu a v ní pomocí Docker kontejnerů, orchestračního nástroje Kubernetes a Spinnakeru implementovat distribuovanou webovou aplikaci. Pro dodržení zásad multi-cloudu bylo využito dvou veřejných poskytovatelů cloudu - Google Cloud Platform a Microsoft Azure, kde byly vytvořeny dva Kubernetes clustery, přičemž každý v odlišném geografickém umístění z důvodu vysoké dostupnosti aplikace.

Pro implementaci byla vybrána HTML webová aplikace postavená na technologii Docker kontejnerů s webovým serverem Nginx. Dále byla provedena instalace všech potřebných nástrojů a následně konfigurace nástroje Spinnaker pro automatické průběžné dodání softwaru.

Poté proběhlo nastavení procesu nasazení ve webovém rozhraní Spinnakeru, které zahrnovalo vytvoření aplikace a čtyř load balancerů pro oba Kubernetes clustery. Nakonec bylo nutné nakonfigurovat dvě pipeline - první s počátečním triggerem pro spuštění při změně Docker image, fází nasazení do testovacího prostředí a manuálním potvrzením od uživatele o tom, zda v nasazení pokračovat. Druhá se skládala z vyhledání Docker image z testovacího prostředí, nasazení aplikace do produkčního prostředí a odstranění clusterů v testovacím prostředí. To vše bylo nutné nastavit pro oba poskytovatele - Google Cloud Platform a Microsoft Azure, kde se Kubernetes clustery nacházely.

Po prvotním manuálním nasazení aplikace bylo několikrát otestováno zmíněné automatické doručení aplikace do produkčního prostředí. Využití automatického průběžného dodání aplikace do prostředí multi-cloudu přineslo požadované výsledky především v podobě ušetřeného času a práce při vývoji a nasazení aplikace do produkčního prostředí. S automatizovaným řešením průběžného dodání již nemusí docházet pouze k nasazování velkých změn v aplikaci, u kterých značně roste riziko chyb nebo k odkládání nasazení nové verze. Každou menší změnu v aplikaci je možné velmi rychle doručit koncovým uživatelům.

Technologie kontejnerů a nástrojů Docker, Kubernetes a Spinnaker je velkým pří-

nosem do budoucna a může změnit celé chápání automatického dodávání a nasazování aplikací do prostředí multi-cloudu. Do dalších let poskytují ty nejlepší předpoklady k úspěchu, tedy širokému praktickému nasazení.

# Zdroje


[1] VELTE, Anthony T., Toby J. VELTE a Robert C. ELSENPETER. *Cloud Computing: praktický průvodce.* Brno: Computer Press, 2011. ISBN 978-80-251-3333-0.

[2] FRAMPTON, Katie. The Differences between IaaS, SaaS and PaaS. *SmartFile* [online]. 2013 [cit. 2017-11-20]. Dostupné z: https://www.smartfile.com/blog/the-differences-between-iaas-saas-and-paas/

[3] MELL, Peter a Timothy GRANCE. *The NIST Definition of Cloud Computing* [online]. 2011 [cit. 2017-11-20]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[4] Legacy Application Transformation. *Mobilize.Net* [online]. [cit. 2017-11-22]. Dostupné z: https://www.mobilize.net/legacy-application-transformation

[5] CONTAINERIZING A LEGACY APPLICATION: AN OVERVIEW. *FP Complete Corp.* [online]. 2017 [cit. 2017-11-22]. Dostupné z: https://www.fpcomplete.com/blog/2017/01/containerize-legacy-app

[6] STINE, Matt. *Migrating to Cloud-Native Application Architectures* [online]. O'Reilly, 2015 [cit. 2017-11-22]. ISBN 978-1-491-92422-8. Dostupné z: https://download3.vmware.com/vmworld/2015/downloads/oreilly-cloud-native-archx.pdf

[7] PETS VS. CATTLE. *Engine Yard* [online]. 2014 [cit. 2017-11-22]. Dostupné z: https://www.engineyard.com/blog/pets-vs-cattle

[8] The History of Pets vs Cattle and How to Use the Analogy Properly. *Cloudscaling* [online]. 2016 [cit. 2017-11-22]. Dostupné z: http://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle/

[9] Pets vs. Cattle: The Elastic Cloud Story. *SlideShare* [online]. 2014 [cit. 2017-11-22]. Dostupné z: https://www.slideshare.net/randybias/pets-vs-cattle-the-elastic-cloud-story

[10] Monolithic Architecture. *Microservices.io* [online]. 2016 [cit. 2017-11-25]. Dostupné z: https://microservices.io/patterns/monolithic.html

[11] BADOLA, Vineet. Microservices architecture: advantages and drawbacks. *Cloud Academy* [online]. 2015 [cit. 2017-11-25]. Dostupné z: https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/

[12] What is Multi-Cloud?. *Rackspace US* [online]. [cit. 2017-11-25]. Dostupné z: https://www.rackspace.com/cloud/multi-cloud

[13] A multi-cloud strategy is the foundation for digital transformation. *O'Reilly* [online]. 2017 [cit. 2017-11-26]. Dostupné z: https://www.oreilly.com/ideas/a-multi-cloud-strategy-is-the-foundation-for-digital-transformation

[14] Multi-Cloud Defined: What Is Multi-Cloud, Exactly?. *BMC Software* [online]. 2017 [cit. 2017-11-26]. Dostupné z: http://www.bmc.com/blogs/multi-cloud-defined-what-is-multi-cloud-exactly/

[15] Top 7 Value Propositions Of A Multi-Cloud Strategy. *Appcara* [online]. [cit. 2017-11-26]. Dostupné z: http://www.appcara.com/blogs/top-7-value-propositions-of-a-multi-cloud-strategy/

[16] Google Cloud Platform Documentation. *Google Cloud* [online]. [cit. 2017-11-28]. Dostupné z: https://cloud.google.com/docs/

[17] Overview of Amazon Web Services: AWS Whitepaper. *Amazon Web Services (AWS)* [online]. 2017 [cit. 2017-11-28]. Dostupné z: https://docs.aws.amazon.com/aws-technical-content/latest/aws-overview/aws-overview.pdf

[18] Co je Azure? *Microsoft* [online]. [cit. 2017-11-28]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-azure/

[19] OpenStack: Welcome Guide. *OpenStack* [online]. 2016 [cit. 2017-11-28]. Dostupné z: https://www.openstack.org/assets/welcome-guide/OpenStackWelcomeGuide.pdf

[20] TURNBULL, James. *The Docker Book: Containerization Is the New Virtualization* [online]. 2014 [cit. 2017-11-30]. ISBN 978-0-9888202-0-3. Dostupné z: http://osgp88fat.bkt.clouddn.com/books/The%20Docker%20Book.pdf

[21] Docker overview | Docker Documentation. *Docker* [online]. [cit. 2017-11-30]. Dostupné z: https://docs.docker.com/engine/docker-overview/

[22] CoreOS is building a container runtime, rkt. *CoreOS* [online]. 2014 [cit. 2017-12-02]. Dostupné z: https://coreos.com/blog/rocket.html

[23] LXD (Linux container hypervisor). *TechTarget* [online]. 2015 [cit. 2017-12-02]. Dostupné z: https://searchitoperations.techtarget.com/definition/LXD-Linux-container-hypervisor

[24] Continuous integration vs. continuous delivery vs. continuous deployment. *Atlassian* [online]. [cit. 2017-12-08]. Dostupné z: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd

[25] Enabling Microservices with Containers & Orchestration – Docker, Mesos, and Kubernetes Explained. *MongoDB* [online]. [cit. 2017-12-08]. Dostupné z: https://www.mongodb.com/containers-and-orchestration-explained

[26] Overview of Docker Compose. *Docker* [online]. [cit. 2017-12-08]. Dostupné z: https://docs.docker.com/compose/overview/

[27] Docker Swarm. *TechTarget* [online]. 2015 [cit. 2017-12-09]. Dostupné z: https://searchitoperations.techtarget.com/definition/Docker-Swarm

[28] Reasons to use Apache Mesos Frameworks. *Container Solutions* [online]. 2015 [cit. 2017-12-09]. Dostupné z: http://container-solutions.com/reasons-use-apache-mesos-frameworks/

[29] POULTON, Nigel. *The Kubernetes Book.* 2018. ISBN 978-1521823637.

[30] Kubernetes Components *Kubernetes* [online]. [cit. 2017-12-09]. Dostupné z: https://kubernetes.io/docs/concepts/overview/components/

[31] Quick Glance at Kubernetes Architectural Building Blocks. *Vitalflux* [online]. 2017 [cit. 2017-12-11]. Dostupné z: https://vitalflux.com/quick-glance-at-kubernetes-architectural-building-blocks/

[32] Cloud Foundry Overview. *Cloud Foundry* [online]. [cit. 2018-01-10]. Dostupné z: https://docs.cloudfoundry.org/concepts/overview.html

[33] Powering and Automating Your Cloud - BOSH. *Cloud Foundry* [online]. [cit. 2018-01-10]. Dostupné z: https://www.cloudfoundry.org/bosh/

[34] PAVLÍK, Jakub. Multi-cloud application orchestration on Mirantis Cloud Platform using Spinnaker. *Mirantis* [online]. 2017 [cit. 2018-01-14]. Dostupné z: https://www.mirantis.com/blog/multi-cloud-application-orchestration-on-mirantis-cloud-platform-using-spinnaker/

[35] Pipelines. *Spinnaker* [online]. [cit. 2018-01-14]. Dostupné z: https://www.spinnaker.io/concepts/pipelines/

[36] Concepts. *Spinnaker* [online]. [cit. 2018-01-14]. Dostupné z: https://www.spinnaker.io/concepts/

[37] Architecture. *Spinnaker* [online]. [cit. 2018-01-14]. Dostupné z: https://www.spinnaker.io/reference/architecture/

[38] How to Set Up Automated Release Analysis in Spinnaker Deployments. *OpsMx* [online]. 2018 [cit. 2018-01-25]. Dostupné z: http://blog.opsmx.com/index.php/2017/06/12/setup-automated-release-analysis-spinnaker/

[39] Containers as a Service (CaaS) as your new platform for application development and operations. *Docker* [online]. 2016 [cit. 2018-01-25]. Dostupné z: https://blog.docker.com/2016/02/containers-as-a-service-caas/

[40] Container Services in the Public Cloud. *Gartner* [online]. 2017 [cit. 2018-01-25]. Dostupné z: https://www.gartner.com/doc/3666417/container-services-public-cloud

[41] Co jsou to webové aplikace a dynamické webové stránky? *Adobe* [online]. [cit. 2018-02-07]. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html

[42] Nginx | Docker Documentation. *Docker* [online]. [cit. 2018-02-07]. Dostupné z: https://docs.docker.com/samples/library/nginx/#exposing-external-port

[43] GitHub For Beginners: Don't Get Scared, Get Started. *ReadWrite* [online]. 2013 [cit. 2018-02-10]. Dostupné z: https://readwrite.com/2013/09/30/understanding-github-a-journey-for-beginners-part-1/

[44] Spinnaker Summit at Netflix. *Armory* [online]. [cit. 2018-02-11]. Dostupné z: http://blog.armory.io/spinnaker-summit-at-netflix/

# Seznam ukázek kódu

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im2-p)

**Podklad pro zadání DIPLOMOVÉ práce studenta**

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Bc. Vašátko Radek | Horní Libchavy 236, Libchavy - Horní Libchavy | I1500751 |

## TÉMA ČESKY:

Analýza a návrh multi-cloud platformy pro geograficky distribuovanou webovou aplikaci

## TÉMA ANGLICKY:

Analysis and design of multi-cloud platform for geographically distributed web applications

## VEDOUCÍ PRÁCE:

Ing. Jakub Pavlík - KIT

## ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout platformu umožňující federaci privátního IaaS cloudu s veřejnými cloudy jako je AWS, DigitalOcean, RackSpace a zjistit tak geografickou dostupnost cloudových aplikací.

1) Úvod do problematiky cloudových řešení
2) Nástroje pro orchestraci microservices
3) Multi-cloud a cluster federation
4) Realizace a výsledky nasazení
5) Závěr

## SEZNAM DOPORUČENÉ LITERATURY:

Hybrid Clouds: The practical solution, Daniel Kusnetzky
Hybrid Cloud Computing - Simple Steps to Win, Insights and Opportunities for Maxing Out Success, Gerard Blokdijk

**Podpis studenta:** ..... **Datum:** .....

**Podpis vedoucího práce:** ..... **Datum:** .....