FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Automatizovaná instalace aplikace pro rozpoznávání řeči v cloudových prostředích

Studijní program: B0613A140005 Informační technologie
Studijní obor: Inteligentní systémy

Autor práce: **Nikita Efimov**
Vedoucí práce: Ing. Ondřej Smola
Ústav informačních technologií a elektroniky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Automatizovaná instalace aplikace pro rozpoznávání řeči v cloudových prostředích

Jméno a příjmení:	**Nikita Efimov**
Osobní číslo:	M20000010
Studijní program:	B0613A140005 Informační technologie
Specializace:	Inteligentní systémy
Zadávající katedra:	Ústav informačních technologií a elektroniky
Akademický rok:	2022/2023

Zásady pro vypracování:

1. Seznamte se s prostředími privátního a veřejného cloudu a platformou Kubernetes.
2. Proveďte rešerši dostupných řešení pro přenositelnou instalaci aplikace.
3. Na základě zvoleného řešení proveďte implementaci a nasazení do vybraného cloudového prostředí.
4. Řešení zdokumentujte a popište postup nasazení do vybraného privátního a veřejného cloudu včetně diskuzí odlišností a nutných kompromisů.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] HOHN, Alan. The Book of Kubernetes. Random House LCC US, 2022. ISBN 1718502648.
[2] MARINESCU, Dan C. Cloud Computing. Morgan Kaufmann Publishers, 2022. ISBN 9780323852777.
[3] MIELL, Ian. Docker in Action. Second Edition. Manning Publications, 2019. ISBN 9781617294808.

Vedoucí práce:	Ing. Ondřej Smola Ústav informačních technologií a elektroniky
Datum zadání práce:	26. října 2022
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 26. října 2022

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

15. března 2023 Nikita Efimov

Automatizovaná instalace aplikace pro rozpoznávání řeči v cloudových prostředích

Abstrakt

Tato bakalářská práce se zaměřuje na automatizaci nasazení aplikace pro rozpoznávání řeči v cloudovém prostředí s využitím systému Kubernetes pro správu kontejnerových aplikací, nástroje Helm pro správu balíčků a Terraform pro správu infrastruktury. Teoretická část práce popisuje tyto technologie. Praktická část představí návrh řešení pro veřejná a privátní cloudová prostředí.

Klíčová slova: cloud, virtualizace, kubernetes, azure, terraform, helm, docker

Abstract

This bachelor thesis focuses on automating the deployment of an application for recognition speech recognition in a cloud environment using Kubernetes for container management applications, Helm for package management and Terraform for infrastructure management. The theoretical part of the thesis describes these technologies. The practical part presents the design of a solution for public and private cloud environments.

Keywords: cloud, virtualization, kubernetes, azure, terraform, helm, docker

Poděkování

Rád bych poděkoval svému vedoucímu bakalářského projektu, Ing. Ondřeji Smolovi, za trpělivost, ochotu a pomoc během práce na tomto projektu. Rád bych také poděkoval společnosti Newton Technologies, pro kterou na tomto projektu pracuji, a zejména Honzovi Václavovi za jeho rady ohledně architektury webové aplikace pro zpracování řeči.

Obsah

Seznam zkratek

k8s	Kubernetes
VM	Virtual Machine
OS	Operating System
IP	Internet Protocol
DNS	Domain Name System
YAML	Yet Another Markup Language
API	Application Programming Interface
IaaC	Infrastructure as a Code
DBaaS	Database as a Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service

Seznam obrázků

Úvod

Cílem této bakalářské práce je navrhnout řešení pro automatizaci nasazení webové aplikace pro rozpoznávání řeči v cloudovém prostředí s využitím platformy Kubernetes. Problém je zajímavý tím, že samotná aplikace se neskládá pouze z jedné komponenty, ale ze sady komponent, přičemž všechny nepoběží pouze na jednom stroji. Automatizace nasazení takové infrastruktury je zajímavá i z hlediska důvěrného seznámení se stále populárnější platformou Kubernetes. V oboru informačních technologií se často objevují nová řešení, platformy a programovací jazyky. Dobrý odborník musí dle mého skromného názoru znát nejnovější populární řešení na trhu a ještě lépe je umět používat. Návrh řešení pro tento projekt úzce souvisí se současnými cloudovými technologiemi, které jsou specifické pro infrastrukturu, jako jsou Kubernetes, Terraform, Docker. V teoretické části se pokusím tyto technologie co nejstručněji popsat a v praktické části předvedu řešení, která jsem přijal pro dosažení cíle tohoto projektu.

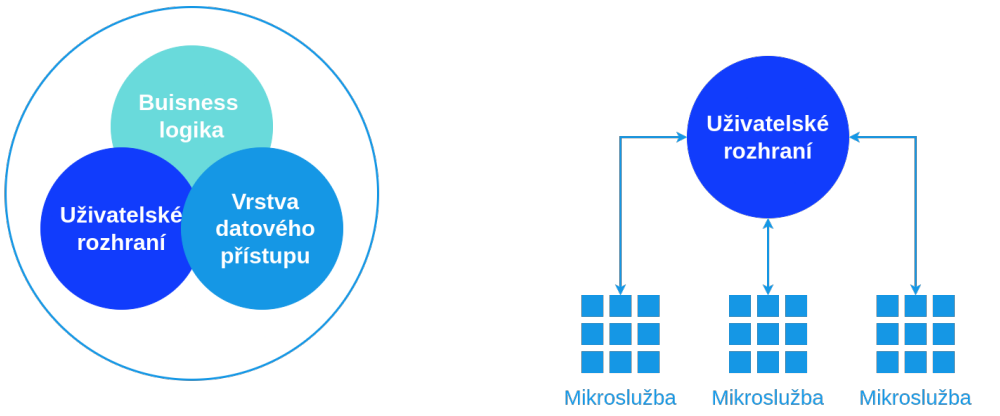
1 Monolitická a mikroservisní architektura

1.1 Co to je mikroservisní a monolitická architektura

Informační technologie prošly za posledních deset let obrovským vývojem. Jednou z nejzajímavějších inovací bylo poskytování služeb přes internet, webové stránky už nejsou jen statickými zdroji textu, ale plnohodnotnými platformami. Prakticky kdokoli s přístupem k internetu může využívat vaše služby odkudkoli na světě.

Vysoká poptávka a příliv zákazníků dříve či později způsobí vývojovému týmu a týmu podpory problémy, jako je například nutné škálování. To se může stát vážným problémem, pokud je software „jediným celým spustitelným souborem“ (záleží na kompilátoru nebo interpretu). Tento přístup se nazývá monolitická architektura nebo jednoduše monolit. Ale co je architektura mikroslužeb? „Model architektury mikroslužeb představuje uspořádání aplikace jako souboru služeb, tzv. mikroslužeb, z nichž každá je dále zodpovědná za určitou část aplikační logiky, obvykle definovanou určitou obchodní schopností.“ (překlad vlastní) [1]

Obrázek 1.1 ukazuje schematickou představu obou typů architektur.



Obr. 1.1: Mikroservisní a monolitická architektura

Zdroj: autor

1.2 Rozdíl mezi mikroservisní a monolitickou architekturou

Hlavní rozdíl mezi oběma architekturami je v tom, jak dobře se škálují. Je-li třeba škálovatelnost, je pro ni nejlépe uzpůsobena architektura mikroslužeb. Proč je výhodnější škálovat mikroslužby než monolit? Připadá-li většina zátěže celého systému na určitou mikroslužbu nebo n-služeb, stačí škálovat pouze je, není třeba škálovat celý systém. Častým problémem monolitů je také to, že chyba v kterékoli části aplikace ovlivní celý systém.

Architektura mikroslužeb samozřejmě není řešením všech problémů a ne všechny programy takový přístup potřebují. Má například následující problémy:

- Komunikace služeb mezi sebou zvyšuje spotřebu sítě, přes kterou tyto služby komunikují, což stojí peníze.
- Je obtížné analyzovat logy, protože každá služba je samostatnou aplikací.
- Potřeba integračních testů se jen zvyšuje, což prodlužuje proces vývoje.

Monolitická architektura by však byla vhodná pro službu, která má jen velmi málo aplikační logiky, jako například časovač Pomodoro, jenž v žádném případě nepotřebuje architekturu mikroslužeb.

V mém případě se jedná o velmi rozsáhlý software, který zahrnuje nejen rozpoznávání řeči z předem připravených zvukových souborů, ale také rozpoznávání zvuku ze streamovaného videa či mikrofonu, identifikaci mluvčího a mnoho dalšího. Tento webový software pro rozpoznávání řeči využívá architekturu mikroslužeb, která rozkládá zátěž systému a urychluje dodávání nových verzí konkrétních služeb, takže není nutné zastavovat celý systém kvůli aktualizaci určité části.

Kromě rozdělení velkého programu na nezávislé služby je třeba všechny tyto mikroslužby izolovat, aby každá z nich byla umístěna ve svém vlastním specifickém prostředí a aby případné chyby v mikroslužbě nemohly ovlivnit celý systém, ve kterém běží ostatní. Tento mechanismus izolace se nazývá kontejnerizace a je vysvětlen v následující kapitole.

2 Kontejnerizace

2.1 Virtualizace

Před definicí kontejnerizace je třeba pochopit, co je to virtualizace. Líbí se mi definice společnosti IBM: „Virtualizace je proces, který umožňuje efektivnější využití fyzického výpočetního hardwaru a je základem cloud computingu.“ (překlad vlastní) [2]

Virtualizace má mnoho podob, ale základní myšlenkou je abstrahování fyzického počítačového hardwaru pro provoz několika virtuálních počítačů (VM), které mají vlastní operační systém (OS) a fungují tak odděleně od stroje, na kterém běží (hostitel).

2.1.1 Hypervisor

„Hypervisor je softwarová vrstva, která koordinuje virtuální počítače.“ (překlad vlastní) [2]

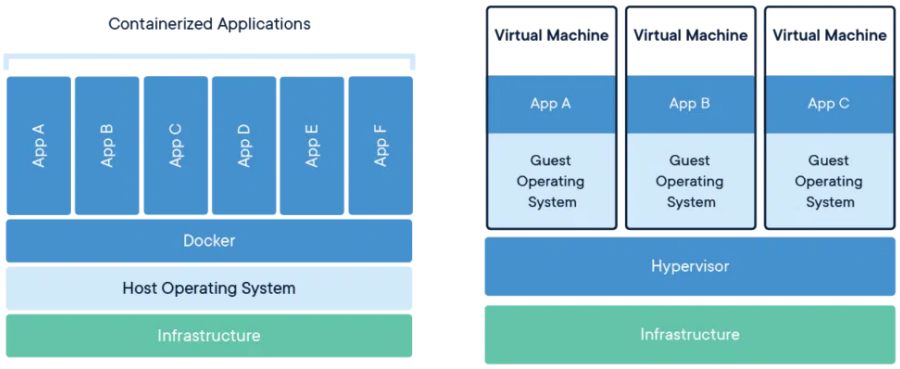
Existují pouze dva typy:

- Hypervisor, který má přímý přístup k hardwaru a nahrazuje tak operační systém.
- Hypervisor, který je aplikací a k přístupu k hardwaru využívá hostitelský OS.

2.2 Definice kontejnerizace

Kontejnerizace je totéž co virtualizace, jen místo vytváření operačního systému pro každý VM používají kontejnery jádro hostitelského operačního systému.

„Kontejnery jsou abstrakcí na aplikační vrstvě, která balí kód a závislosti dohromady“ (překlad vlastní) [3], zatímco virtuální počítač je hardwarová abstrakce. Kontejnery jsou tedy mnohem lehčí než virtuální počítače a jejich nasazení je rychlejší.



Obr. 2.1: Rozdíl mezi architekturou kontejnerů a virtuálních strojů

Zdroj: [3]

2.2.1 Jmenný prostor kontejnerů neboli izolace

Kontejnery jsou izolovány pomocí jmenného prostoru neboli namespace. Prostor jmen je součástí jádra Linuxu a slouží k oddělení prostředků jádra tak, aby určitá skupina procesů měla přístup pouze k prostředkům v prostoru jmen, do kterého patří. Jinými slovy, prostor jmen je způsob, jak izolovat procesy. Kontejnerům jsou například automaticky přiděleny následující jmenné prostory:

- pid – identifikátor procesu,
- mount – seznam připojených bodů (adresářů) pro připojování souborových systémů,
- ipc – mechanismus izolace paměti,
- network – informace o síťové komunikaci.

2.3 Docker

„Docker poskytuje způsob, jak zabalit a distribuovat aplikace jako lehké a přenosné kontejnery, které lze spustit na jakékoli platformě. To umožňuje snadnější a rychlejší nasazení aplikací v různých prostředích, od notebooku vývojáře až po produkční server.“ (překlad vlastní) [4]

Mnoho lidí je zmateno, když se jich někdo zeptá, co je to Docker. Zdá se totiž, že kontejnery jsou to samé co Docker, ale není to tak. Proto bych rád doplnil citát na začátku této podkapitoly tím, že Docker zaručuje implementaci standardů OCI (Open Container Initiative). V souladu s těmito standardy lze kontejnery provozovat v různých prostředích. Docker se tedy nerovná kontejnerům, Docker je platforma.

Samotný Docker kontejnery nespouští, za to je zodpovědný *containerd*: „Standardní běhové prostředí pro kontejnery s důrazem na jednoduchost, robustnost a přenositelnost.“ (překlad vlastní) [5]

„Docker obraz je lehký, samostatný spustitelný balíček, který obsahuje vše potřebné ke spuštění programu, včetně kódu, běhového prostředí, knihoven, proměnných prostředí a konfiguračních souborů. Obrazy se vytvářejí ze souboru Dockerfile, který specifikuje instrukce pro vytvoření obrazu. Jakmile je obraz sestaven, lze jej odeslat do registru a poté jej mohou ostatní vytáhnout a spustit aplikaci v kontejneru.“ (překlad vlastní) [4]. Příkladem registru, kam lze ukládat kontejnery, je Docker Hub.

Izolace kontejnerů a odlehčení oproti VM jsou skvělé pro architekturu mikroslužeb. Problém nastává s nárůstem mikroslužeb v aplikaci, a tím i v počtu kontejnerů, které je třeba udržovat. Údržbou mám na mysli následující body:

- provoz kontejnerů na několika strojích,
- konfigurace a vyrovnávání zátěže sítě,
- plánování kontejnerů na stroj se zajištěním splnění požadavků na zdroje,
- automatické restartování v případě chyby.

Naštěstí jsou k dispozici nástroje, které tyto a další problémy řeší. Takovým nástrojům se říká kontejnerové orchestrátory a nejpopulárnějším řešením na trhu je Kubernetes.

3 Kubernetes

„Kubernetes poskytuje způsob správy kontejnerových aplikací v distribuovaném prostředí a nabízí automatizované nasazení, škálování a správu kontejnerových aplikací. Umožňuje vývojářům soustředit se na psaní kódu spíše než na správu infrastruktury a zároveň poskytuje provozním týmům nástroje, které potřebují k efektivní správě základní infrastruktury. Kubernetes usnadňuje nasazování a správu kontejnerizovaných aplikací ve velkém měřítku, poskytuje způsob, jak automatizovat proces nasazování, škálování a správy kontejnerů, a usnadňuje zajištění toho, aby aplikace fungovaly podle očekávání.“ (překlad vlastní) [6]

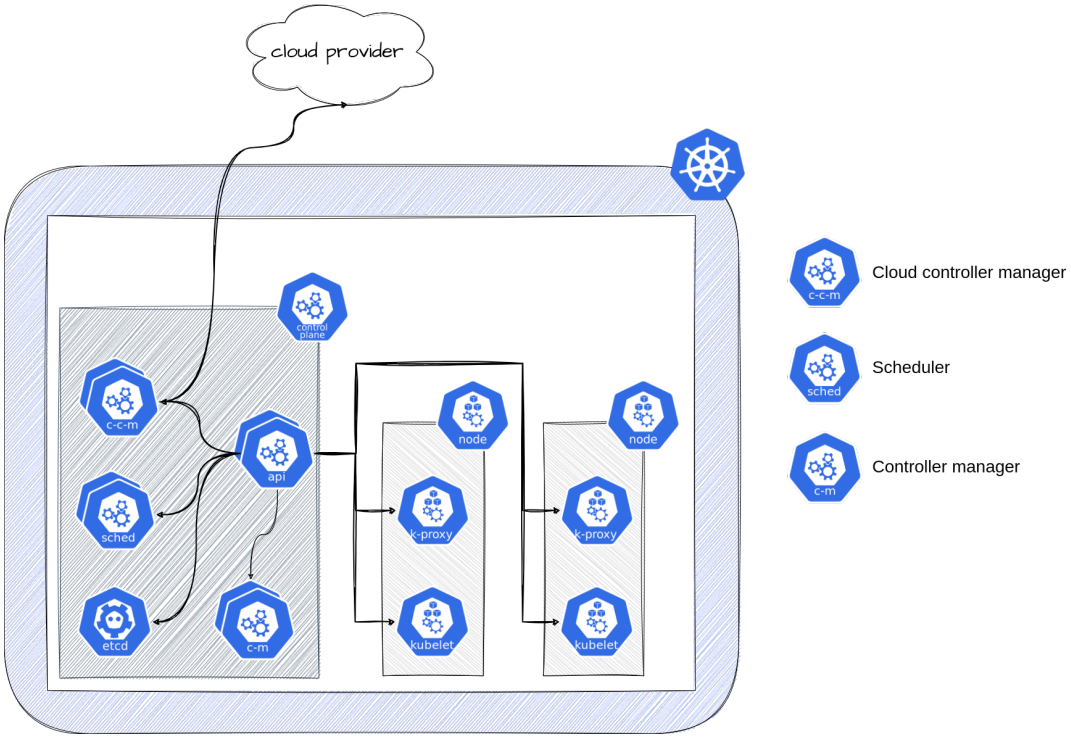
Kubernetes poskytuje:

- Vyrovnávání zátěže sítě mezi více službami.
- Orchestraci úložiště – možnost dynamicky připojovat úložiště.
- Automatické nasazení a zpětné vrácení – Kubernetes přivádí aplikaci do požadovaného stavu a také poskytuje možnost vrácení zpět.
- Samočinné obnovení nebo restartování kontejnerů.

Systém Kubernetes není dodáván s předpřipravenými řešeními například pro orchestraci úložiště. Namísto toho poskytuje rozhraní API, lze tedy sestavit svůj cluster jako konstruktér. Kubernetes má však základní komponenty.

3.1 Architektura a komponenty Kubernetes.

Cluster Kubernetes se skládá ze dvou kategorií serverů: Master a Worker. Servery se nazývají uzly (Nodes).



Obr. 3.1: Komponenty Kubernetes

Zdroj: autor podle [7]

Control plane je sada komponent, které spravují všechny uzly v clusteru a obvykle jsou na samostatném stroji.

kube-apiserver	Klientská část správy Kubernetes. Kontroluje příchozí konfiguraci, může také vytvářet konfiguraci z dotazů a poskytovat informace o komponentách clusteru i o samotném clusteru.
etcd	Distribuované úložiště dat typu klíč-hodnota používané jako hlavní úložiště pro všechna data clusteru Kubernetes.
kube-scheduler	Plánuje, na kterém uzlu budou kontejnery nasazeny.
kube-controller-manager	Spouští procesy pro řízení clusteru. Například upozornění na selhání stroje.
cloud-controller-manager	Je stejný jako *kube-controller-manager*, ale je určen pro poskytovatele cloudu. Například při selhání stroje sleduje, zda byl z cloudu odstraněn.

Tabulka 3.1: Komponenty Control Plane

Komponenty uzlu jsou spuštěny na každém uzlu v clusteru, které podporuje běhové prostředí Kubernetes.

kubelet	Zajišťuje, aby byly spuštěny všechny kontejnery.
kube-proxy	Konfiguruje síťová pravidla v uzlech.
container runtime	prostředí pro spouštění kontejnerů.

Tabulka 3.2: Komponenty uzlů

Kubernetes lze rozšiřovat pomocí rozšíření. Doplňky jsou volitelné, ale jeden z nich je povinný, a to interní systém doménových jmen (DNS) clusteru. Rozšíření jsou umístěna do jmenného prostoru kube-system, protože pokrývají celý cluster. Vzhledem k tomu, že Kubernetes je open source, komunita pro něj vytvořila mnoho doplňků. Je možné převzít kontrolu nad sítí, logováním, monitorováním a lze také napsat vlastní operátor v Kubernetes pro správu budoucích zdrojů v clusteru, pokud současná nabídka neřeší nějaký úzce zaměřený problém.

3.2 Objekty Kubernetes

3.2.1 Základní objekty

Pod	Základní objekt Kubernetes, což je sada jmenných prostorů Linuxu. Zjednodušeně řečeno se jedná o abstrakci, která spojuje kontejnery dohromady. Pod je nejmenší nasaditelná jednotka. Pody se často restartují a jsou automaticky vytvářeny jinými objekty.
ReplicaSet	Slouží k udržování určitého počtu podů replik. Při vytváření jsou zadány labely, aby bylo možné pody v clusteru snadno najít.
Deployment	Další vrstvou abstrakce, ve které je předepsán požadovaný stav, který pak kontrolér Deployment (jeden ze správců kube-control--managers) sleduje a udržuje.
StatefulSet	Objekt pro práci se stavovými aplikacemi, které potřebují zajistit konzistenci dat.
Service	Konfiguruje síť pro pody. Service má statickou IP a umožňuje používat interní DNS.
Ingress	Slouží k přístupu ke clusteru zvenčí.

Tabulka 3.3: Základní objekty Kubernetes

3.2.2 Úložiště

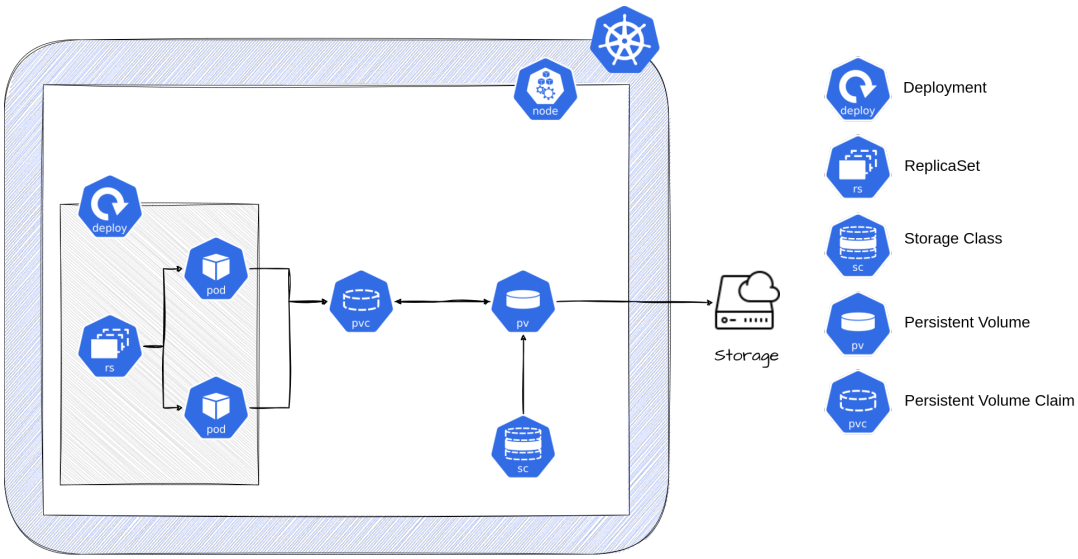
Pod může obsahovat více kontejnerů a tyto kontejnery mají možnost ukládat data do podu. Problém je v tom, že pod je prostředek, který podle návrhu může být a s největší pravděpodobností bude restartován, jinými slovy znovu vytvořen, čímž se ztratí všechna data. Některé aplikace potřebují zachovat data navzdory restartům a Kubernetes používá objekt PersistentVolume pro trvalé uložení dat.

PersistentVolume (PV) je abstrakce definující způsob poskytování úložiště a jeho použití. Základní nastavení jsou:

- velikost,
- typ přístupu – čtení nebo čtení a zápis pro jeden nebo více uzlů,
- odkaz na šablonu dotazu.

Šablona požadavku je PersistentVolumeClaim(PVC), která určí, kolik paměti nebo jednotek CPU je požadováno a jaký typ přístupu, a tuto šablonu pak použije modul pod. Pokud PVC vyhovuje PV, pak se na něj naváže.

PV lze vytvářet ručně, ale objekt StorageClass umožňuje dynamické vytváření PV, když to PVC vyžaduje.



Obr. 3.2: Příklad poskytování úložiště v Kubernetes

Zdroj: autor

3.2.3 Tajemství a konfigurace pro aplikace

Kubernetes používá ConfigMap k ukládání necitlivých dat. Název objektu naznačuje, že se jedná o konfiguraci aplikace. Pokud změníte ConfigMap, komponenta kubelet si toho všimne a aktualizuje konfiguraci, ale není to tak jednoznačné. Hodnoty v ConfigMap lze použít jako proměnné prostředí nebo namontovat celý soubor do podu stejným mechanismem jako úložiště a nutná je opatrnost. Pokud namountujete ConfigMap do cesty v adresáři *'app/configs'*, očekává se, že se vytvoří konfigurační soubor, což je pravda, ale s jedním problémem: vše v adresáři configs zmizí a k dispozici bude pouze namountovaný soubor. Abyste se tomu vyhnuli, musíte soubor připojit v cestě s názvem souboru, tedy *'app/configs/mycfg.yaml'*, ale pak kubelet změny konfigurace neuvidí a nebude je aktualizovat.

Secret je objekt, který uchovává citlivá data, pro která se používá šifrování. Ve výchozím nastavení je Secret uložen nešifrovaně v komponentě etcd a kdokoli s přístupem k API clusteru si může Secret vyžádat a upravit.

3.3 Síťování v Kubernetes

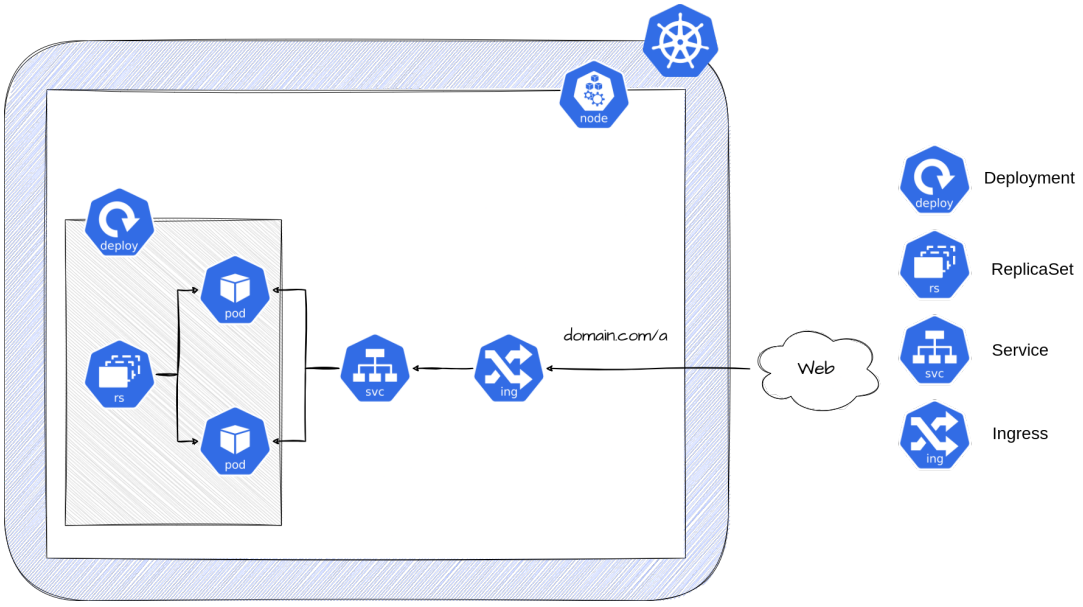
Pody v Kubernetes získají dynamickou IP adresu, která se po opětovném vytvoření změní. V tomto prostředí je velmi obtížné nastavit síť v rámci clusteru. Řešením tohoto problému je objekt Service, který je zodpovědný za směrování provozu do podů, které jsou s touto službou spojeny pomocí selektoru labelů.

Objekt Service může být vázán na několik podů a přebírá úlohu vyrovnávání provozu mezi nimi. Dělá to velmi jednoduše: při požadavku DNS (název servisu v konfiguraci je doména) Service volá všechny pody a vybírá IP v závislosti na proxy-módu, ale tento výběr je ve výchozím nastavení náhodný. Service může být několika typů, a to:

- ClusterIP - standardní typ, připojení pouze v rámci clusteru. Vytvoří statickou virtuální IP pro pody a automaticky vyrovnává provoz vázaných podů.
- NodePort – vytvoří statický port na všech uzlech v clusteru a nasměruje provoz na připojené pody, čímž je zvenčí zpřístupní clusteru.
- LoadBalancer - využívá load balancer v clusteru k vyrovnávání sítí ze všech uzlů a pody se stanou dostupnými mimo cluster.
- headless - vytvoří názvy DNS pro příslušnou sadu podů. Každý z těchto záznamů vede na IP podu.

Přestože lze objekt Service použít k přijímání přenosů z internetu, je lepší jej k tomuto účelu nepoužívat. Idiomatické by bylo použít objekt Ingress. „Ingress je prvek Kubernetes, který umožňuje definovat pravidla pro směrování externích přenosů HTTP a HTTPS do interních služeb v clusteru. Je to v podstatě sada pravidel, která vám umožní vystavit službu HTTP/S běžící v clusteru internetu tím, že nakonfigurujete externí load balancer, aby na ni směroval provoz na základě cest URL nebo názvů domén.“ (překlad vlastní) [8]

Na obrázku 3.3 je ukázka toho, co se děje v clusteru se správně nakonfigurovaným Deploymentem, Service a Ingress. Deployment vytvoří pody, pomocí label selectoru budou propojeny Servisy s pody a Ingress nasměruje externí webový provoz. Důležitým detailem je, že ne všechny pody musí mít Service.



Obr. 3.3: Příklad síťování v Kubernetes

Zdroj: autor

3.4 Jak nasazovat v systému Kubernetes

Aby bylo možné aplikaci nasadit v systému Kubernetes, musí být kontejnerizovaná. Pro nasazení těchto kontejnerů v Kuberentes je potřebné popsat, které objekty se k tomu používají. Samotné objekty jsou popsány pomocí manifestů. Manifest je konfigurace zapsaná v jazyce YAML, která popisuje požadovaný stav. Jedná se o deklarativní programovací paradigma, to znamená, že není zapotřebí popisovat nebo programovat vytváření těchto objektů, Kubernetes to udělá sám. Pro použití konfigurace je nezbytné nainstalovat nástroj kubectl, který umožňuje přístup k rozhraní API platformy.

Příkaz k použití konfigurace manifestu z souboru s názvem *test.yaml*

```
1  $ kubectl apply -f test.yaml
```

4 Cloud Computing

Vzhledem k tomu, že cloudové technologie přímo souvisí s mou bakalářskou prací, považuji za nutné je zmínit podrobněji, než se dostanu k praktické části.

4.1 Definice

Pojem cloud lze definovat jako propojené servery tvořící distribuční systém, ale pan Marinescu má na tento pojem jiný názor: „Cloud computing je model, který umožňuje všudypřítomný, pohodlný síťový přístup na vyžádání ke sdílenému fondu konfigurovatelných výpočetních zdrojů, které lze rychle poskytnout a uvolnit s minimálním úsilím o správu nebo interakci s poskytovatelem služby.“ (překlad vlastní) [9]. Obě tyto definice jsou správné a lze je sloučit, ale definice pana Marinescu ukazuje, proč je cloud zajímavý zejména pro společnosti podnikající v oblasti IT.

Hlavní výhodou cloudové technologie je, že místo nákupu vlastních zdrojů a zajišťování péče o ně může společnost snadno zakoupit plně spravované zdroje od poskytovatele cloudu. To je výhodné, protože tím odpadá zátěž spojená s údržbou vlastní infrastruktury.

4.2 Typy cloudů

4.2.1 Veřejný

Veřejný cloud je typ cloudu, kde jsou všechny součásti infrastruktury a služby řízeny třetí stranou, konkrétně samotným poskytovatelem cloudu. Jedná se o nejobecnější případ cloudu a nejlevnější, protože správa a údržba je na poskytovateli.

4.2.2 Privátní

Na rozdíl od veřejného, privátní znamená, že zdroje vlastní jedna konkrétní společnost. Může je také vlastnit třetí strana, která službu poskytuje, ale takovým způsobem, že jsou určeny pouze pro jednu společnost.

Důvodem vzniku takového typu je zabezpečení dat společnosti. Běžným případem je, že interní infrastruktura je umístěna za firewallem společnosti, a navíc privátní cloud umožňuje přizpůsobit prostředky podle svých potřeb a zajistit tak soulad se specifickými požadavky na ukládání dat, jako je například GDPR.

4.2.3 Hybridní

Hybridní cloud je kombinací obou výše uvedených možností. Obvykle jsou procesy rozděleny tak, že ty, které pracují s citlivými daty, jsou v privátní části a ostatní ve veřejné části.

4.2.4 Porovnání

Výhodou veřejného cloudu oproti jiným typům je cena. Řízení nákladů v cloudu je obrovský problém, protože je nutné detailně pochopit, jaký výpočetní výkon přesně potřebujete, abyste nepřišli o další peníze, což se často stává. Nevýhodou je, že data a zdroje jsou plně pod kontrolou poskytovatele.

Privátní cloud má zjevnou výhodu, a to bezpečnost citlivých dat a konfigurovatelnost zdrojů. Nevýhodou jsou náklady, ale soukromí za to stojí.

Hybridní cloud má výhody i nevýhody obou typů, ale není lepší ani horší, je to prostě kombinace obou.

4.3 Typy cloudových služeb

Cloudová služba je služba, platforma nebo program nabízený a vlastněný poskytovatelem cloudu. Typy služeb se liší podle toho, nakolik pro vás poskytovatel již nějakou část vyřešil. Tou částí myslím virtualizaci, operační systém, runtime, sítě, úložiště, servery.

U každého typu napíšu, co může uživatel spravovat, což znamená, že o zbytek se postará poskytovatel.

4.3.1 Infrastruktura jako služba

Infrastruktura jako služba (anglická zkratka IaaS) znamená, že poskytovatel nabízí možnost vybudovat si vlastní infrastrukturu.

Uživatel může spravovat operační systém, běhové prostředí, aplikace a data.

4.3.2 Platforma jako služba

Platforma jako služba (anglická zkratka PaaS) znamená, že poskytovatel nabízí vše potřebné pro provoz aplikace uživatele a ukládání dat.

Uživatel může spravovat aplikace a data.

4.3.3 Software jako služba

Software jako služba (anglická zkratka SaaS) znamená, že poskytovatel nabízí plně spravovaný program. Například Google Workspace nebo Microsoft 365.

Uživatel nic nespravuje.

4.4 Způsoby poskytování služeb

Mnou zmíněné vyšší typy služeb musí být nějakým způsobem poskytovány – způsobů je několik.

4.4.1 Webový portál

Jedná se o webové uživatelské rozhraní, kde si uživatel může kliknout a přečíst si informace o službě (službách) a podle pokynů ji nastavit.

Podle mého názoru je výhodou pouze to, že uživatel má k dispozici vizuálně příjemné grafické rozhraní pro seznámení se s poskytovatelem služby.

Nevýhodou je, že poskytování služeb nelze automatizovat.

4.4.2 Příkazová řádka

Příkazová řádka je nástrojem komunikujícím s API poskytovateli cloudu. Obvykle se jedná o proprietární řešení od konkrétního poskytovatele cloudu. Všechny příkazy pro poskytování jsou uvedeny v dokumentaci.

Výhodou je flexibilita, protože má uživatel možnost definovat ve skriptu vše podle potřeby, a také to, že je možné automatizovat poskytování.

Nevýhodou je to, že se prakticky nedá udržovat aktuální stav služeb nebo infrastruktury. To je problém, protože nejsou-li sledované a spravované stavy, nelze se vrátit zpět, pokud se po aktualizaci něco zhroutí.

4.4.3 Infrastruktura jako kód

Nástroje pro infrastrukturu jako kód (anglická zkratka IaaC) jsou nástroje, které mohou poskytovat služby a infrastrukturu psaním kódu. Všechny tyto nástroje obvykle využívají deklarativní programovací proces pro poskytování, to znamená, že uživatel popíše očekávaný stav a nástroj tento stav poskytne prostřednictvím komunikace s rozhraním API poskytovatele cloudu.

Výhodou je monitorování a správa stavu a také snadná automatizace, protože kód je mnohem přehlednější než rozsáhlé skripty.

Jedinou nevýhodou je podle mého názoru to, že tyto nástroje je třeba ve srovnání se skripty a webovými portály nastudovat.

5 Praktická část

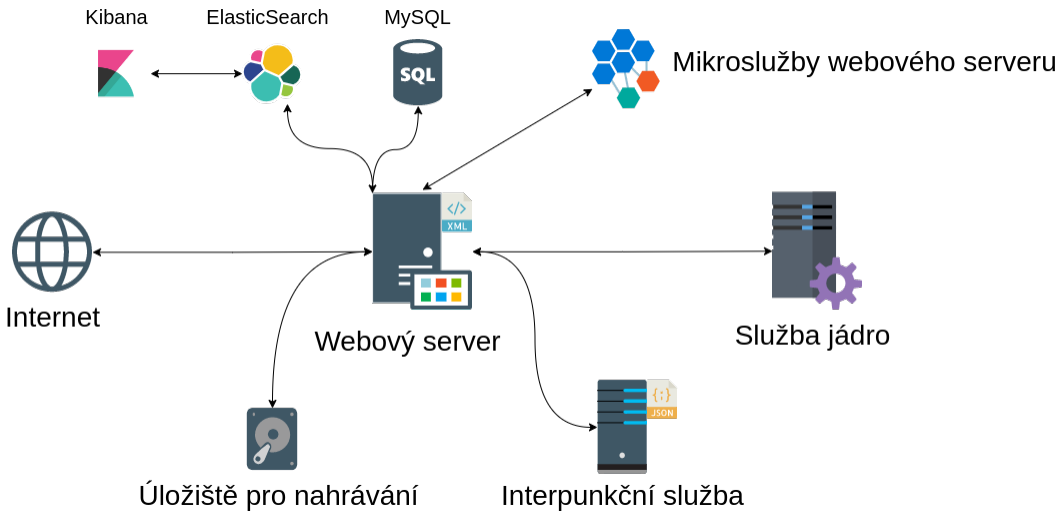
Hlavním cílem této bakalářské práce je vytvořit automatizované přenosné nasazení aplikace do cloudu. V této části bude ukázána implementace řešení pro privátní i veřejný cloud s využitím moderních technologií Kubernetes a Terraform. Řešení bude vycházet z analýzy samotné aplikace a následného návrhu.

5.1 Seznam použitých balíčků a nástrojů

Níže je uveden seznam použitých nástrojů a balíčků a jejich verze, aby bylo zřejmé, na čem projekt běží.

Název	**Verze**
docker engine	23.0.1
containerd	1.6.19
runc	1.1.4
docker-init	0.19.0
kubectl	1.25.2
k3d	5.4.6
k3s	1.24.4-k3s1
helm	3.9.4
terraform	1.4.2
azure-cli, azure-cli-core	2.46.0
azure-cli-telemetry	1.0.8

5.2 Analýza a návrh



Obr. 5.1: Architektura aplikace

Zdroj: autor

Obrázek 5.1 ukazuje architekturu aplikace pro rozpoznávání řeči, pro kterou budu navrhovat cloudové instalační řešení.

Popis komponentů aplikace:

- Dvě databáze: MySQL a Elasticsearch. Pro práci s Elasticsearch je zapotřebí dodat Kibanu.
- Hlavní webový server s webovým uživatelským rozhraním. Vyžaduje ukládání souborů a konfigurační soubor.
- Jádro služby, na kterém se plánují a spouštějí modely neuronových sítí pro rozpoznávání řeči.
- Interpunkční služba - samostatná služba pro opravu interpunkce po rozpoznání. Vyžaduje konfigurační soubor.
- Mikroslužby webového serveru. Název mluví sám za sebe, je důležité si uvědomit, že webový server používá vlastní proxy službu pro směrování provozu na mikroslužby.

Jak jsem již zmínil, „aby bylo možné aplikaci nasadit v systému Kubernetes, musí být kontejnerizovaná.“ 3.4. Naštěstí má již každá komponenta, se kterou musím pracovat, vytvořený obraz Dockeru. Proto potřebuji přenést architekturu z obrázku 5.1 do clusteru Kubernetes, respektive popíšu manifesty, které vytvoří objekty Kubernetes. Předtím bych ale rád zmínil problém databází v distribučních systémech a chystaná řešení.

5.2.1 Databáze v cloudu

Podle článku zveřejněného na blogu Google Cloud může být nasazení databází v prostředí distribuovaných systémů, včetně Kubernetes, problematické. „Přes veškerý růst na aplikační vrstvě se datová vrstva díky kontejnerizaci tolik neprosadila. To není překvapivé, protože kontejnerizovaná pracovní zátěž musí být ze své podstaty odolná vůči restartům, škálování, virtualizaci a dalším omezením. Takže manipulace s věcmi, jako je stav (databáze), dostupnost pro ostatní vrstvy aplikace a redundance databáze, může mít velmi specifické požadavky. To činí provozování databáze v distribuovaném prostředí náročným.“ (překlad vlastní) [10]. To v žádném případě neznamená, že je to nemožné. Existuje mnoho databází, které dobře fungují v distribuovaných systémech - Cassandra, MongoDB a Elasticsearch jsou dobrými příklady.

Přestože MySQL nebyla navržena pro distribuované systémy, lze ji provozovat na více serverech a jsou k dispozici šablony, které to umožňují. Standardní přístup spočívá ve spuštění několika databází MySQL tak, že jedna z nich se stane hlavní (v angličtině se často označuje jako Leader, Master, Primary) a provádí operace čtení a zápisu, a ostatní repliky (Worker, Slave, Secondary, Standby) mohou pouze číst. Důvodem, proč může zapisovat pouze jedna instance, je zachování konzistence dat, protože provedlo-li by více replik operaci se stejnými daty, dostal by koncový uživatel při čtení irelevantní výsledky.

Nejzajímavější je, když master a workery ztratí vzájemný kontakt, například když dojde k chybě hardwaru na master serveru. Workerům pak není jasné, zda je master skutečně vypnutý, a je teoreticky možné, že stále přijímá požadavky uživatelů.

Pro řešení takových situací zřizují třetí strany v systému Kubernetes operátory. Problémem je, že pro databázi Postgre existuje několik operátorů, které jsou aktivně podporovány open source komunitou, zatímco MySQL není tak bohatá a kromě operátoru Oracle není příliš na výběr.

Pokud může být provozování databází v distribuovaných systémech problém, jaké řešení bych měl zvolit?

Jak je uvedeno v článku [10], existují tři možné scénáře pro databáze:

- plně spravovaná databáze poskytovatelem cloudu,
- spuštění databáze mimo cluster Kubernetes,
- spuštění databáze v Kubernetes pomocí operátora.

Protože podle čtvrtého bodu zadání této bakalářské práce musí být řešení pro privátní i veřejný cloud, bylo po konzultaci s vedoucím přijato následující rozhodnutí. Výsledek musí realizovat dva z výše uvedených bodů tak, že u veřejného clusteru bude na výběr mezi plně spravovaným MySQL nebo nasazením databází v Kubernetes, v privátním cloudu půjde o nasazení buď mimo cluster, nebo uvnitř clusteru.

Plně spravovaný Elasticsearch ve veřejném cloudu může být drahé řešení, ale nemá smysl, protože Elasticsearch má řešení pro nasazení v Kubernetes.

Nasazení MySQL databáze v systému Kubernetes v rámci tohoto projektu je ukázkou toho, že je to možné, spíše než řešením připraveným k produkci.

5.3 Přenosné nasazení do Kubernetes

Jak jsem zmínil v kapitole 3.4, pro nasazení infrastruktury v Kubernetes je nutné napsat manifesty. Po napsání manifestů vyvstávají velmi zajímavé otázky: „Jak je mám distribuovat?“ a „Jak je předat uživateli, aby mohl infrastrukturu nasadit u sebe?“ Jedním řešením by bylo poslat mu všechny soubory s dokumentací k nasazení, ale problém nastane, když klient potřebuje změnit určitý parametr v některých nebo všech manifestech kvůli specifikům svého clusteru. Pro řešení tohoto problému existují nástroje, které popíšu níže a vyberu ten nejvhodnější pro svůj úkol.

5.3.1 Skaffold

Nástroj, který nevyžaduje žádné konfigurační soubory YAML pro Kubernetes, místo toho je Skaffold sám vytváří z konfiguračních souborů Dockeru. Skaffold sleduje zdrojový kód a jakmile jsou aplikovány změny, Skaffold sám vytvoří kontejner Docker a nasadí jej do clusteru Kubernetes.

Výhody: Skvělé pro ladění a rychlé testování, jak se mikroslužba bude chovat v clusteru.

Nevýhody: Platí pouze pro vývoj. Chcete-li nastavit síť mezi pody nebo nasadit v clusteru jiný objekt, který není součástí zdrojového kódu aplikace, musíte jej ještě zvlášť popsat v YAML.

5.3.2 Kustomize

Nástroj umožňující provádět změny na popsaných objektech. Provádí se pomocí vytvoření souboru YAML, který specifikuje nová pole a jejich hodnoty nebo pole změněná novými hodnotami, přičemž název objektu, který má být změněn, by se měl přesně shodovat, aby Kustomize pochopil, na co se má daná konfigurace aplikovat.

Výhody: Kustomize je od verze 1.14 zabudován do nástroje kubectl. Vhodné pro malé změny.

Nevýhody: Velmi snadno způsobí zmatek a ztratíte přehled o tom, co se má změnit, co ne a kde. Není to přenosné řešení, protože stále musíte upravovat již napsané soubory YAML pro Kustomize, abyste konfiguraci přizpůsobili svým potřebám.

5.3.3 Helm

Správce šablon a balíčků pro Kubernetes. Umožňuje šablonovat všechny konfigurační soubory YAML a upravovat je pomocí jediného souboru.

Koncepty Helm:

- Templates - umožňují vytvářet dynamické manifesty Kubernetes.
- Charts - balíček je složkou konfiguračních souborů YAML.
- Release - instance balíčku, která byla nainstalována do clusteru.

- Revision - aktualizace a změny, které se v daném vydání vyskytnou. Jinými slovy, verzování, které lze použít k vrácení zpět.
- Values - hodnoty, které jsou použity šablonou.
- Repositories - úložiště pro ukládání a distribuci balíčků.

Výhody: Snadná distribuce snadno přizpůsobitelných manifestů Kubernetes. Snadná aktualizace, úprava a instalace objektů Kubernetes.

Nevýhody: Subjektivní, ale zaznamenal jsem nevýhodu v tom, že výsledný konfigurační soubor pro celý cluster je neuvěřitelně velký, což samozřejmě závisí na tom, jak velký je projekt.

5.3.4 Volba řešení

Pomocí aplikace Skaffold je možné vytvářet manifesty automaticky ze zdrojového kódu a následně je kopírovat, což však neumožňuje žádnou flexibilitu při přizpůsobování manifestů. Kustomize je dobré řešení pro malý počet manifestů, ale pokud konfiguraci Kustomize přizpůsobujete svému clusteru, musíte vše měnit ručně. Helm je skvělým řešením pro vytváření dynamických šablon, které lze přizpůsobit vašim potřebám, a nabízí dokonce i distribuční řešení. Proto jsem si vybral Helm.

5.4 Plán nasazení do veřejného cloudu

Mám za úkol poskytnout rekonfigurovatelnou konfiguraci, kde lze zvolit buď použití DBaaS služba plně udržované databáze, nebo nasazení databází v Kubernetes. Tato konfigurace je možná pouze pomocí nástroje, který umožňuje popsat celou infrastrukturu v kódu a následně ji nasadit. Pro tento projekt budu používat nejoblíbenější řešení na trhu, kterým je Terraform. „Terraform Cloud umožňuje automatizaci infrastruktury pro poskytování, dodržování a správu jakéhokoli cloudu, datového centra a služby," je uvedeno na domovské stránce společnosti HashiCorp (překlad vlastní) [11]

Pro nasazení ve veřejném cloudu je nutné:

- Vybrat potřebné služby a komponenty infrastruktury od poskytovatele cloudu.
- Popsat tyto součásti a služby pomocí Terraformu.

Samozřejmě musím si také vybrat poskytovatele cloudu, u kterého infrastrukturu nasadím. Po konzultaci s vedoucím projektu byl jako poskytovatel cloudu vybrán Microsoft Azure, protože vedoucí má s tímto konkrétním členem veřejného cloudu největší zkušenosti, a může tak snadno a rychle ověřit, zda mnou navržená infrastruktura dává smysl.

Tímto končí kapitola o analýze a návrhu. Dalším krokem je příprava lokálního clusteru Kubernetes pro testování manifestů a simulaci prostředí privátního cloudu.

5.5 Příprava lokálního clusteru Kubernetes

Kubernetes je vynikající platforma pro orchestraci kontejnerů, jejíž velkou nevýhodou je velmi nízký vstupní práh. Kromě složitého učení se používání samotné platformy je výzvou také její nasazení na serverech, protože kromě nasazení samotného clusteru je nutno zajistit i správné řešení úložiště, což může být pro nezkušeného inženýra náročný úkol. Chystám se nasadit cluster Kubernetes lokálně na notebooku a jednou z klíčových částí instalace clusteru je zakázání odkládací paměti a úplné naformátování disků, zejména proto, že samotný plnohodnotný cluster Kubernetes nemá na notebooku velký smysl. Proto existuje odlehčená verze Kubernetes nazvaná *K3s*, která slouží k místnímu testování a spouštění málo zatížených úloh.

Pro testování manifestů na mém notebooku mám zájem o možnost spustit cluster s více uzly. K dosažení tohoto cíle mám na výběr ze dvou možností. První z nich je vytvořenímněkolika virtuálních počítačů a nasazení clusteru na nich, což vyžaduje mnoho prostředků. Druhou možností je nasadit cluster v Dockeru, kde by kontejnery byly uzly. To je pro mě skvělá možnost, protože, jak jsem psal v kapitole 2, kontejnery jsou jednodušší než virtuální stroje. Otázkou je, jak se mi podaří nasadit cluster v Dockeru. Snadnou odpovědí na ni je *k3d* utilita, která nasadí cluster k3s v Dockeru, proto koncovka d.

Chtěl bych upřesnit, že v žádném případě nehodlám používat *k3d* v produkčním prostředí, pouze jako prostředí pro rychlé testování konfigurace. Kubernetes v Dockeru totiž znamená virtualizaci ve virtualizaci a v produkci to bude mít vliv na rychlost zejména diskových operací zápisu a čtení.

Před vytvořením clusteru je nutné nainstalovat Docker, *k3d* a nástroj *kubectl*, což lze snadno provést podle jejich oficiální dokumentace.

Po instalaci všech nástrojů zbývá spustit příkaz pro inicializaci clusteru se dvěma uzly:

Inicializace clusteru

```
1  $ k3d cluster create dev -agents 2
```

Kontrola uzlů

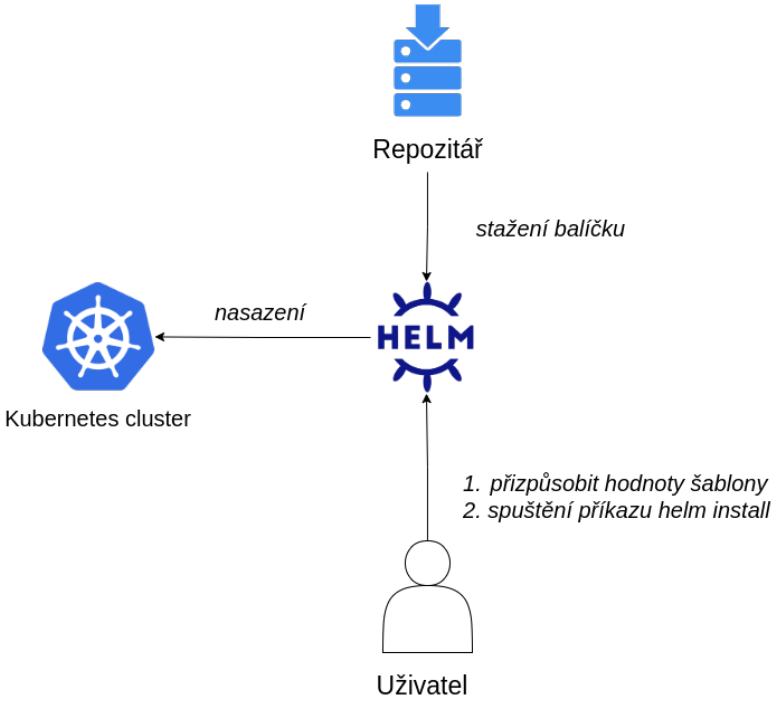
```
1  $ kubectl get nodes
```

Ve sloupci *STATUS* musí mít každý uzel stav *Ready*.

Lokální cluster Kubernetes pro testování manifestů je připraven. Dalším krokem je napsání manifestů.

5.6 Práce s Helmem

Než se pustím do práce, je dobré se seznámit s tím, jak práce s Helm probíhá. Obrázek 5.2 ukazuje jednotlivé kroky. V podstatě by takto měl vypadat postup při použití mého řešení.



Obr. 5.2: Diagram nasazení pomocí Helmu

Zdroj: autor

Pro nasazení aplikace v clusteru Kubernetes stačí změnit hlavní konfigurační soubor *values.yaml* nebo použít standardní hodnoty a spustit jediný příkaz. Helm odešle požadavek na rozhraní API Kubernetes a v případě neplatné konfigurace vrátí chybu nebo informaci, že konfigurace byly zkontrolovány a nyní je Kubernetes nasadí.

5.6.1 Struktura balíčků Helm

Balíčky Helm mají svou vlastní speciální strukturu. Spuštěním příkazu *helm create <název>* se vytvoří následující struktura:

- *charts*, adresář pro uložení dalších balíčků Helm nebo adresář pro závislosti projektu,
- *Chart.yaml* - soubor popisující balík závislostí, verzi, název, popis,
- *templates* - adresář pro uložení šablon tohoto projektu,
- *templates/_helpers.toml* - pomocníci šablon, aby se neopakovala stejná šablona a používala se jako funkce,
- *values.yaml* - hlavní konfigurační soubor hodnot pro šablonu projektu.

5.6.2 Závislosti

Pro tento projekt mám následující seznam závislostí:

```
dependencies:
- condition: traefik.enabled
  name: traefik
  repository: https://helm.traefik.io/traefik
  version: 20.4.x
- name: common
  repository: https://charts.bitnami.com/bitnami
  version: 2.2.x
- name: ng-speech
  repository: oci://registry-1.docker.io/newtontechnologies
  version: 2.0.x
```

- *traefik* je Ingress pro přístup k webovému serveru z externího clusteru.
- *common* je sada užitečných funkcí pro šablonování.
- *ng-speech* je služba jádro.

5.6.3 Psaní šablon

Protože šablony některých komponent z 5.1 jsou rozsáhlé, místo uvádění kódu popíšu, jaké jsem použil objekty Kubernetes pro jednotlivé komponenty.

Již dříve jsem zmínil, že pro Kubernetes se vytváří speciální objekty Operator 5.2.1, které slouží ke správě databází. Tyto objekty jsou potřebné především pro automatickou konfiguraci databázového clusteru.

Databázový cluster je nutný především pro škálování operací čtení a dosažení vysoké dostupnosti v případě havárie jedné instance, aby se nezastavila činnost celého databázového systému.

Důležitou součástí šablon pro databáze bude boolean *true* nebo *false*, aby bylo možné některé z databází ponechat v clusteru Kubernetes nenasazené.

Bohužel, protože Operatory vyžadují při inicializaci obor názvů, který lze zadat pouze při spuštění příkazu install, musí být Operatory nainstalovány před vytvořením databázového clusteru. Dalším důvodem je to, že nechci míchat všechny tyto komponenty 5.1 v jednom jmenném prostoru.

5.6.4 Webový server

Obr. 5.3: Objekty Kubernetes pro webový server

Zdroj: autor

Z komponent 5.3 zobrazených na obrázku bych podrobněji vysvětlil komponenty ConfigMap a Job.

ConfigMap

Existují dva způsoby, jak přidat konfigurační soubor do podu, ConfigMap a namountovat soubor. Druhý způsob důrazně nedoporučuji, protože soubor bude muset být připojen z hostitele, a tak při restartování podu na jiném hostiteli nebude soubor k dispozici.

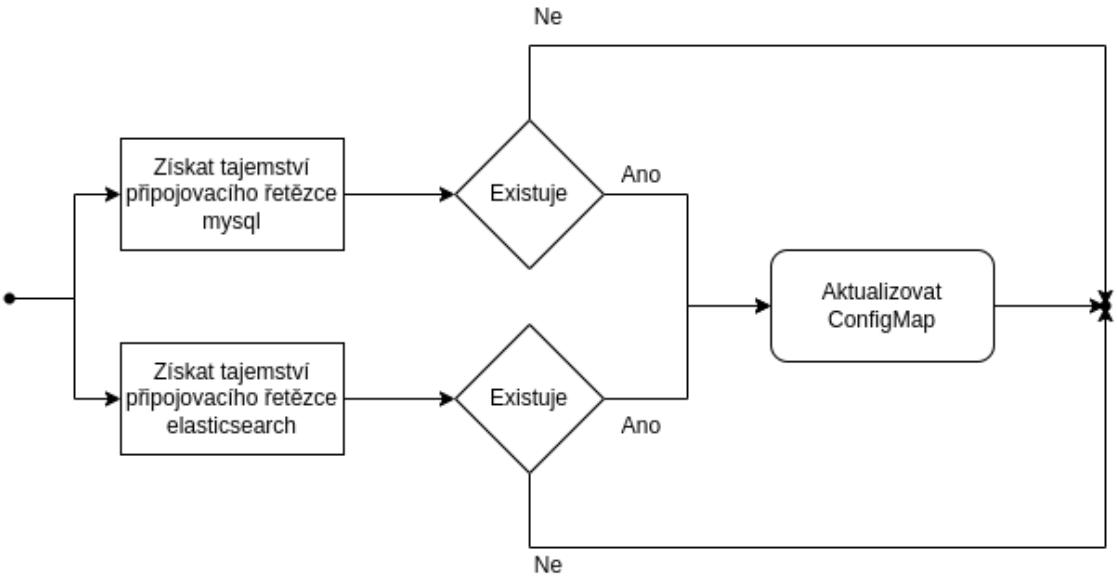
Proto jsem zvolil následující řešení. Konfigurací bude ConfigMap, kde klíčem je název souboru a hodnotou obsah souboru. Aby byly změny v konfiguraci rozpoznány, přidám do podu anotaci v podobě hashe souboru, což znamená, že pokud je hash jiný, pak byl soubor změněn a pod se znovu vytvoří s novou konfigurací.

Job

Aby se hlavní webový server mohl připojit k databázím, musí být parametry připojení nastaveny v konfiguračním souboru. To je problém, protože pro lepší zabezpečení používám automaticky generovaná hesla, která nejsou předem známa. Proto jsem se rozhodl vytvořit objekt Job, jehož účelem bude převzít hesla MySQL

a ElasticSearch a aktualizovat konfigurační soubor hlavního webového serveru pomocí nástroje *sed*.

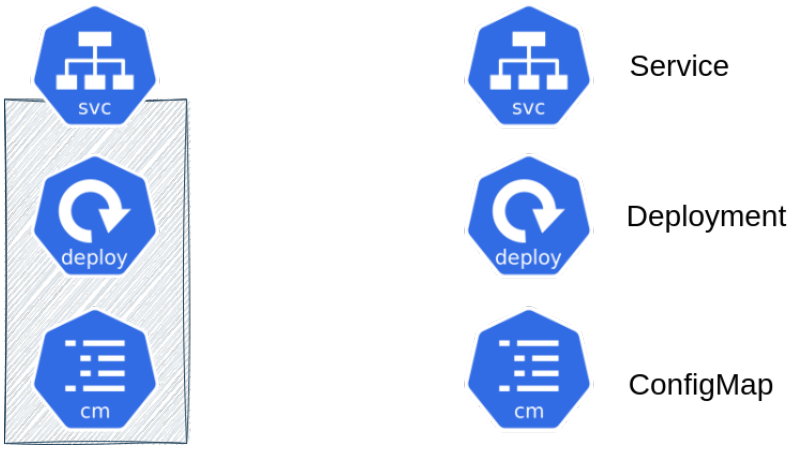
Kubernetes ve výchozím nastavení neumožňuje podům provádět v clusteru žádné změny. Protože Job musí mít možnost měnit ConfigMap a získávat objekty Secret, definoval jsem ServiceAccount a RoleBinding, ve kterých jsem povolil pouze operace *get* pro objekty Secret a *get*, *create*, *update* pro ConfigMap. Diagram 5.4 ukazuje proces aktualizace ConfigMap.



Obr. 5.4: Diagram aktualizace ConfigMap

Zdroj: autor

5.6.5 Interpunkční server

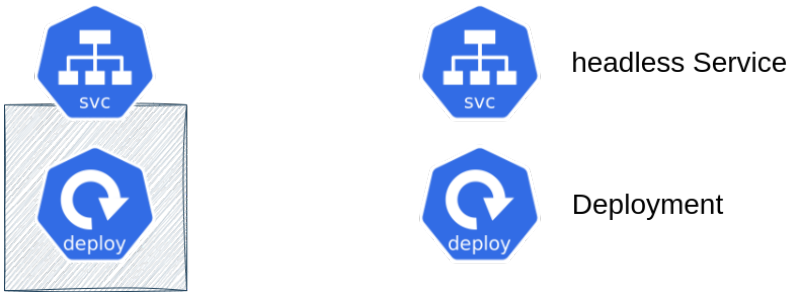


Obr. 5.5: Objekty Kubernetes pro interpunkční server

Zdroj: autor

Interpunkční server používá stejný konfigurační soubor jako webový server, ale na rozdíl od webového serveru nepotřebuje žádné aktualizace. Jedná se o typ konfigurace, který se nastavuje pouze jednou.

5.6.6 Mikroslužby



Obr. 5.6: Objekty Kubernetes pro mikroslužby

Zdroj: autor

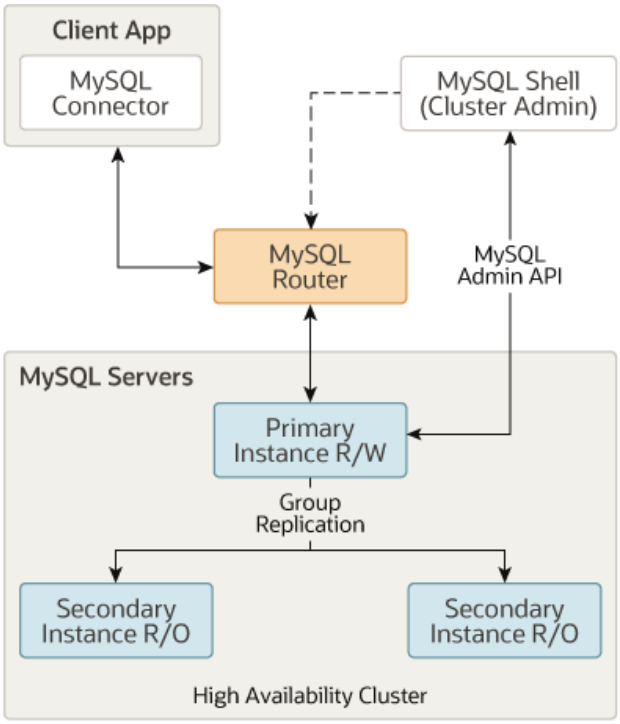
Jedná se o nejběžnější typ nasazení kontejnerů v systému Kubernetes, konkrétně o jednoduché Deployment plus Service. Není však tak standardní, protože webový server 5.3 pro přístup k mikroslužbám má vlastní proxy, z tohoto důvodu musí být objekt Service headless, protože tento proxy nemůže používat virtuální IP adresu vytvořenou pro výchozí objekt Service 3.3.

V tomto projektu mám zatím k dispozici pouze jednu mikroslužbu.

5.6.7 MySQL

Pro MySQL použiji MySQL Operator od společnosti Oracle, který mi umožňuje nasadit InnoDB cluster v Kubernetes.

InnoDB je mechanismus ukládání dat. Proč vůbec potřebujete řešení pro ukládání dat specifické pro MySQL? MySQL je relační databáze, která používá transakce s vlastnostmi ACID (Atomicity, Consistency, Isolation a Durability). Tento návrh není vhodný pro distribuované systémy s paralelními operacemi, které mohou pravidlo konzistence dat snadno obejít. InnoDB je navržena tak, aby MySQL v distribuovaných systémech mohla stále zaručovat principy ACID.



Obr. 5.7: Schéma InnoDBCluster

Zdroj: [12]

Obrázek 5.7 zobrazuje architekturu *InnoDBCluster*, která obsahuje:

- *Primary Instance* - hlavní instance serveru, která může číst a zapisovat.
- *Secondary Instance* - pracovní servery, které mohou pouze číst.
- *MySQL Router* - cluster load balancer, který je zodpovědný za přesměrování provozu.

Process instalace Operatoru:

Přidání repozitáře mysql-operator

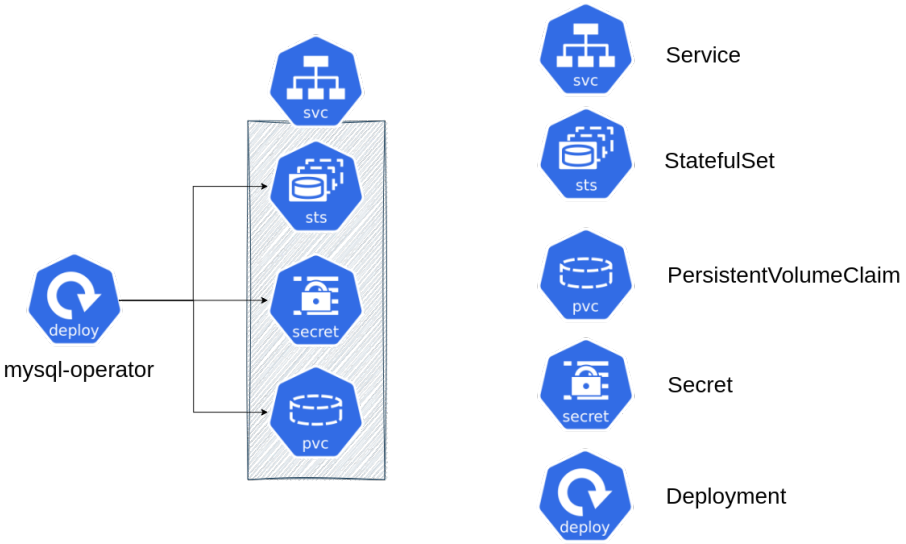
```
$ helm repo add mysql-operator https://mysql.github.io/mysql-operator/
```

Nastavení Operatoru do jmenného prostoru mysql-operator

```
$ helm install mysql-operator mysql-operator/mysql-operator --namespace mysql-operator --create-namespace
```

Následně jsem popsal manifest pro Secret s údaji o připojení a pro cluster InnoDB podle dokumentace [13] a pak je šablonoval.

Na obrázku 5.8 je vidět, jak vypadá cluster z obrázku 5.7 v Kuberentes. Operator sám vytváří další komponenty podle potřeby.



Obr. 5.8: Objekty Kubernetes pro InnoDBCluster

Zdroj: autor

Důležitá poznámka je, že mé řešení nezahrnuje automatické zálohování, protože nemohu předem vědět, kam je uložit. Zálohy musí být v odděleném úložišti od clusteru Kubernetes. Bohužel řešení migrace databází do Kubernetes je stále diskutabilní, takže mé řešení není ideální.

5.6.8 Elasticsearch a Kibana

Pro Elasticsearch bylo vybráno řešení Elastic On Cloud, které poskytuje možnost nasadit cluster Elastic včetně Kibany jako balíček.

Postup instalace Operatoru je následující:

Přidání repozitáře elastic

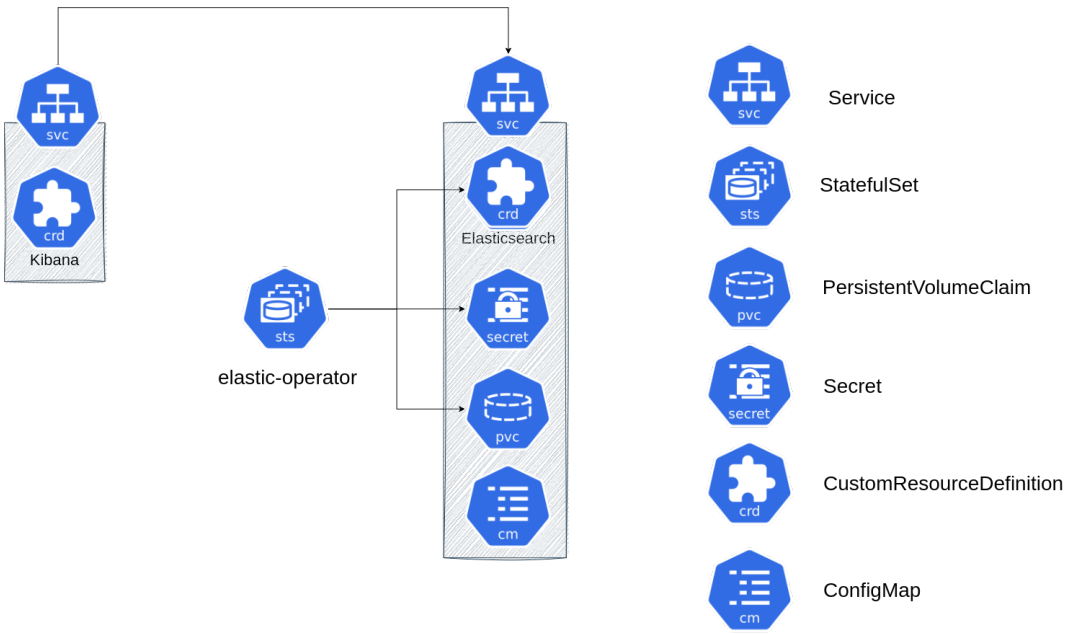
```
$ helm repo add elastic https://Helm.elastic.co
```

Nastavení Operatoru v jmenném prostoru elastic-system

```
$ helm install elastic-operator elastic/eck-operator -n
    elastic-system --create-namespace
```

Poté stačí popsat manifesty pro Kibanu a Elasticsearch podle dokumentace [14] a šablonovat je.

Obrázek 5.9 ukazuje, jak se nasazený Elasticsearch a Kibana zobrazují v systému Kubernetes.



Obr. 5.9: Objekty Kubernetes pro Elasticsearch a Kibana

Zdroj: autor

5.6.9 Přístup z internetu

Pro přístup z internetu jsem použil řešení Ingress od společnosti traefik, ve kterém jsem zadal, ke kterým službám chci poskytnout přístup zvenčí. Traefik není třeba předem nějak instalovat, je přidán jako závislost 5.6.2. Pro tento projekt stačí zajistit přístup zvenčí pouze pro webový server. V případě nasazení ve veřejném cloudu je třeba zadat IP adresu interního load balanceru, aby Ingress fungoval a směroval provoz.

5.6.10 Distribuce

K distribuci balíčku Helmů existuje „nepřeberné množství možností, pokud jde o hostování vlastního repozitáře“ (překlad vlastní) [15]. Je to proto, že úložištěm balíčku může být libovolný webový server reagující na požadavek HTTP GET. Rozhodl jsem se umístit balíček do centra Docker Hub. Na první pohled se zdá zvláštní, že balíček Kubernetes a obrazy Dockeru mohou být umístěny na stejném místě, ale je to proto, že obrazy Dockeru a balíčky Helm jsou uloženy v tar archivech.

Postup pro umístění balíčku Helm do centra Docker Hub:

Vytvoření balíčku Helm z hotového projektu

```
$ helm package <name>
```

Přihlásit se pomocí nástroje docker

```
$ docker login
```

Odeslání balíčku Helm do Docker Hubu

```
$ helm push <name>-0.0.1.tgz  oci://registry-1.docker.io
  /<repository>
```

Pokud má úložiště osobní přístupový token PAT, je třeba jeho použití autorizovat následujícím způsobem, například předchozím nastavením PAT do proměnné REG_PAT.

```
$ echo $REG_PAT | helm registry login registry-1.docker.
  io -u <username> --password-stdin
```

5.7 Nasazení v privátním cloudu

V této fázi je již k dispozici vše, co je potřeba k nasazení architektury z obrázku 5.1 v privátním cloudu. Nemohu automatizovat samotné nasazení clusteru Kubernetes, protože tento proces vyžaduje vlastní přístup, a proto musí být pro použití mého řešení splněny následující požadavky:

- Mít nasazený cluster Kubernetes s hotovým řešením úložiště a StorageClass.
- Nainstalovat nástroj Helm.

Postup nasazení se v zásadě neliší od postupu 5.2, pouze je potřeba přidat nastavení Operatoru. Pokyny pro nasazení jsou proto následující:

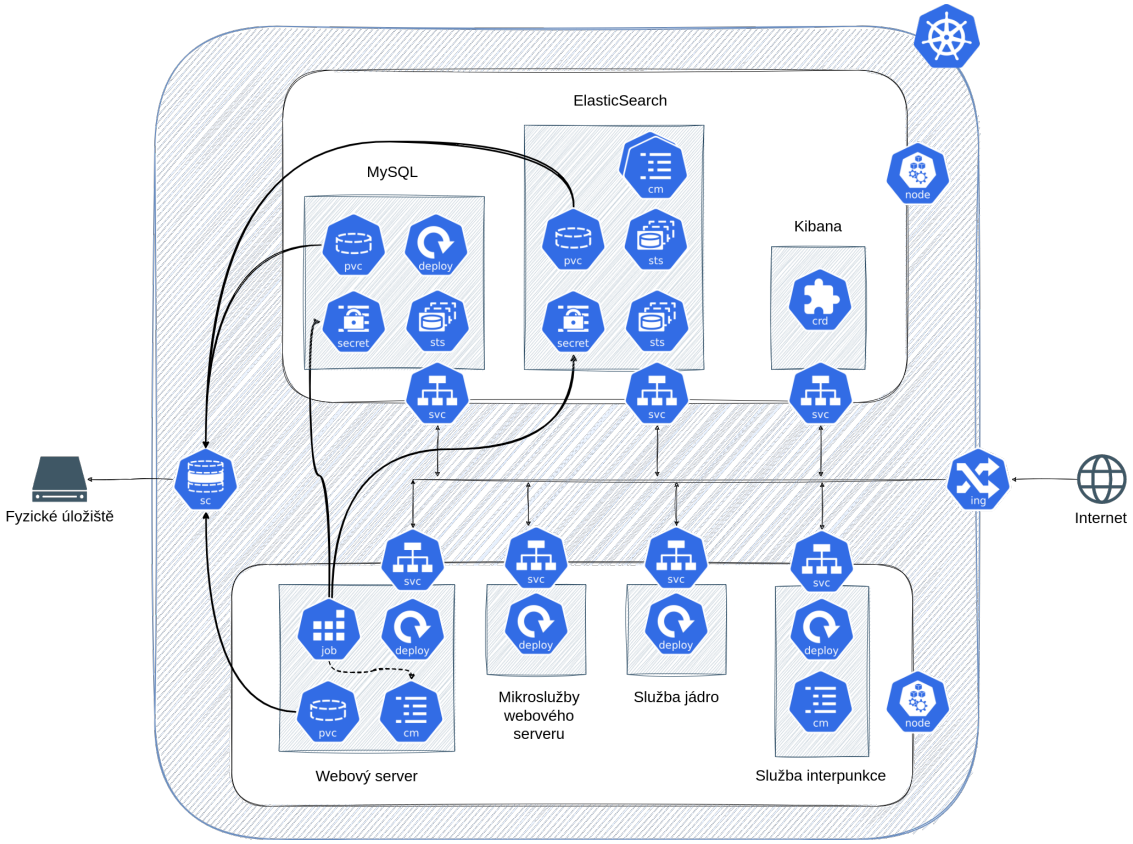
- Změnit hodnoty šablon v souboru values.yaml.
- Nastavit databázové operátory pomocí 5.6.7 a 5.6.8.
- Spustit příkaz pro nasazení pomocí Helmu:

```
  $ helm upgrade --install <name> <helm_package> \
--set-file scribe.config.scribeSettingsXML=<path> \
--set-file spp.config.sppSetupJson=<path> \
--set-file scribe.config.fakeng2storeJson=<path>
```

Kde -set-file slouží k vytvoření mapy ConfigMap z lokálního souboru.

O třetím konfiguračním souboru jsem se nezmínil, protože se jedná o hotfix a je nutný pouze pro kompatibilitu se staršími verzemi služby jádra.

Po nasazení vypadá cluster následovně:



Obr. 5.10: Nasazená aplikace v systému Kubernetes

Zdroj: autor

Výše uvedený diagram ukazuje, jak vypadá cluster s výchozími hodnotami. Namísto kreslení jednotlivých podů jsem je zabalil do objektů Deployment a StatefuSet tak, jak budou ve skutečnosti v clusteru. Každý z těchto Deployment a StatefuSet má objekt Service pro vnitřní komunikaci v rámci clusteru a ty, které potřebují úložiště, mají požadavek PersistentVolumeClaim zpracovávaný pomocí StorageClass.

V této fázi lze řešení pro privátní cloud považovat za dokončené. Dalším krokem je vytvoření řešení pro veřejný cloud Microsoft Azure.

5.8 Microsoft Azure

Podle průzkumu společnosti Synergy Research Group je Microsoft Azure druhým nejpopulárnějším poskytovatelem cloudových služeb na světě [16]. Každý poskytovatel cloudu má vlastní přístup k seskupování prostředků. V Azure musí mít všechny prostředky vlastní skupinu prostředků - objekt, který sdružuje uživatelem vybrané prvky jeho infrastruktury.

5.8.1 Výběr potřebných služeb Azure

Microsoft Azure je cloudová platforma, která poskytuje širokou škálu služeb. Širokou nabídkou mám na mysli více než dvě stovky služeb, z nichž budu muset pro tento projekt vybrat ty nejvhodnější.

Pro návrh dobrého řešení tohoto projektu není třeba se učit všechny služby, protože samotné řešení se skládá z velmi logických a triviálních komponent.

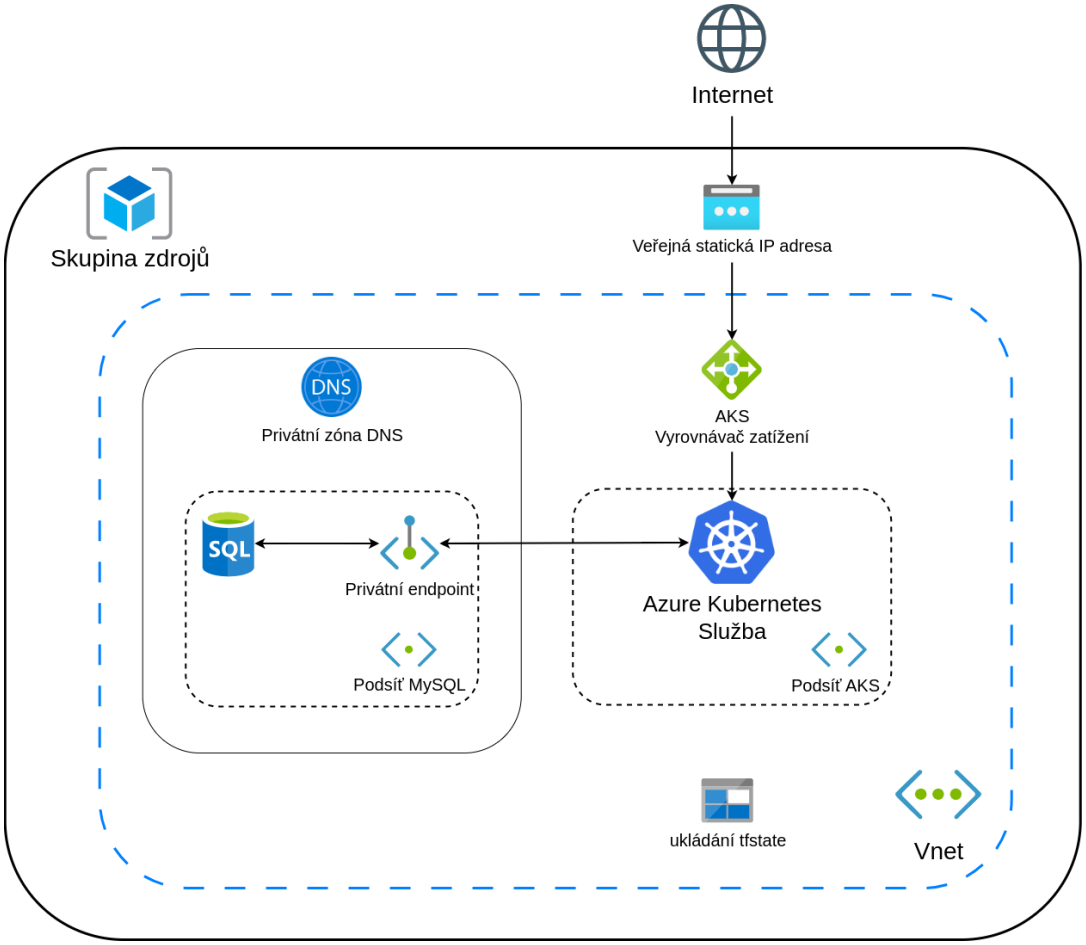
Před výběrem komponentů musím mít představu o tom, jaký by měl být výsledek. Potřebuji nasadit cluster Kubernetes a databáze MySQL. Jedna ze služeb clusteru Kubernetes by měla být přístupná z internetu a komunikace mezi databázemi by měla probíhat v rámci privátní sítě.

Protože Elasticsearch lze provozovat v prostředí distribuovaného systému bez problému, zůstane v clusteru Kubernetes.

Na základě toho uvádím seznam služeb, které jsem vybral:

- Virtual Network – potřebná k hostování dalších služeb, které spolu mají komunikovat prostřednictvím sítě.
- Azure Kubernetes Service - plně obsluhovaný cluster Kubernetes v Azure.
- Azure MySQL Flexible Server - plně udržovatelná databáze MySQL.
- Private DNS - potřebný k nalezení MySQL ve virtuální síti.
- Public Static IP - pro přístup k webovému serveru z internetu, který je v clusteru Kubernetes.
- Subnet - podsítě pro Kubernetes a MySQL.

V Azure je pro vytvoření jakékoli komponenty nutné vytvořit Resource Group neboli skupinu prostředků, která by měla sdružovat logicky související komponenty.



Obr. 5.11: Nasazená infrastruktura v Azure

Zdroj: autor

Obrázek 5.11 ukazuje vybudovanou infrastrukturu v Microsoft Azure.

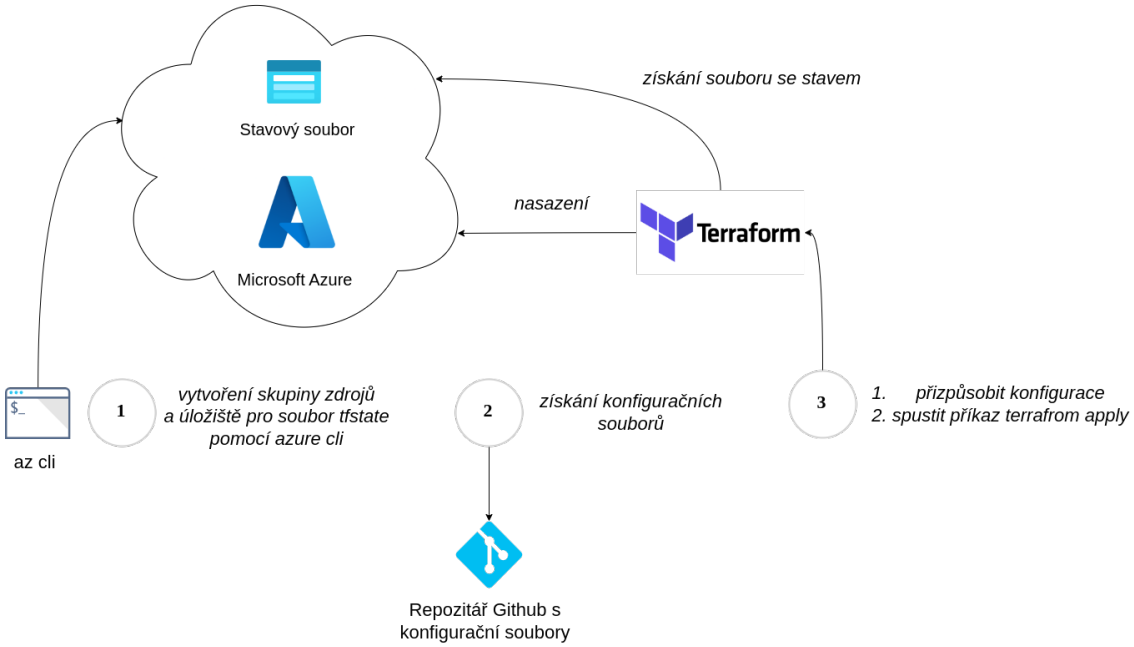
5.9 Práce s Terraformem

Jak jsem psal v 5.2, budu k nasazení infrastruktury v Azure používat nástroj Terraform; níže popíšu postup.

Terraform jako nástroj IaaC používá deklarativní programovací paradigma, což znamená, že programátor popíše požadovaný stav a Terraform použije API poskytovatele cloudu k implementaci tohoto stavu.

Aby bylo dosaženo konzistence stavu při změnách infrastruktur různými osobami a aby se zabránilo vytváření celých infrastruktur od začátku při malých změnách, ukládá Terraform aktuální stav infrastruktury do souboru *tfstate*. V případě infrastruktury v cloudu je logické, že by se tento soubor měl nacházet přímo v cloudu, a nikoli na lokální pracovní stanici, ze které byla infrastruktura nasazena.

Pro lepší představu uvádím diagram práce s Terraformem na obrázku 5.12.



Obr. 5.12: Diagram nasazení pomocí terraformu

Zdroj: autor

Před vytvořením infrastrukturních komponent musím předem vytvořit skupinu prostředků a úložiště pro soubor *tfstate*. Kvůli omezením Terraformu v něm nemohu tento proces popsat, ale namísto toho napíšu krátký skript, který použije nástroj *az cli* pro správu Azure, který vytvoří skupinu prostředků a úložiště. Skript poté informuje uživatele o tom, jak tuto komponentu přidat do Terraformu.

Pro každou z komponent jsem vytvořil adresář představující samostatný modul se strukturou:

- *main.tf* - konfigurace komponenty
- *variables.tf* - hodnoty pro komponentu nastavené v main.tf v kořenovém adresáři projektu
- *outputs.tf* - hodnoty exportované tímto modelem, například pro použití jinou komponentou.

Tímto způsobem se mi podařilo vytvořit komunikaci a závislost mezi komponentami, což znamená, že virtuální síť bude vytvořena dříve, než se Terraform pokusí vytvořit cluster Kubernetes.

5.10 Nasazení ve veřejném cloudu

Pro nasazení ve veřejném cloudu Azure pomocí mého řešení je nutné nejprve splnit následující požadavky:

- Mít účet Azure.
- Nainstalovat nástroj az cli.
- Nainstalovat Terraform.

Postup nasazení je totožný jako v diagramu5.12. Výsledkem nasazení je infrastruktura zobrazená na obrázku 5.11.

Terraform má vynikající příkaz *terraform destroy*, který zajistí, že všechny komponenty vytvořené pomocí Terraformu budou zničeny. To je užitečné, abyste nemuseli plýtvat penězi na testování, protože žádný poskytovatel cloudových služeb neposkytuje své služby zdarma.

Stejně jako při nasazení do Kubernetes stačí několik kroků. V Terraformu můžete také měnit hodnoty zdrojů, které jsem se rozhodl oddělit do samostatného souboru *main.tf* umístěného v kořenovém adresáři, aby byl proces konfigurace co nejpodobnější procesu konfigurace Helmu.

Pro nasazení samotné aplikace v Kubernetes se používají stejné Helm šablony jako pro nasazení v privátním cloudu, což naznačuje, že je návrh řešení přenositelný.

5.11 Testování

V této kapitole bych rád ukázal, jak jsem testoval, zda je vše správně nakonfigurováno. Všechny konfigurace jsou lokální, proto vynechám krok stahování balíčku.

5.11.1 Prostředí privátního cloudu

Především chci zakázat plánování podů na hlavním uzlu, k3d to nemá ve výchozím nastavení zapnuté.

Nastavení limitu NoSchedule v hlavním uzlu

```
$ kubectl taint nodes k3d-dev-server-0 node-role=not-
    eligible:NoSchedule
```

Chci rozdělit nasazení do dvou uzlů. Na prvním bude databáze a na druhém vše ostatní. Abych toho dosáhl, potřebuji na uzlech nastavit štítky, podle nichž pak budu plánovat pody.

Nastavení štítků na uzlech

```
$ kubectl label nodes k3d-dev-agent-0 databases=yes
$ kubectl label nodes k3d-dev-agent-1 databases=no
```

Kromě plánování podů na různých uzlech chci také rozdělit pomocí jmenných prostorů. Databáze budou v *databases* a zbytek v *default*.

Instalace mysql-operator

```
$ helm install mysql-operator mysql-operator/mysql-
    operator -n databases --create-namespace
```

Instalace elasticsearch-operator

```
$ helm install elasticsearch-operator elastic/eck-
    operator -n databases --create-namespace
```

V konfiguračním souboru *values.yaml* jsem nastavil parametr *affinity* pro každou komponentu. Parametr *affinity* se používá pro plánování. V tomto případě jsem pro databáze stanovil podmínku, že pokud je hodnota řetězce databases rovna yes, pak budu plánovat na tomto uzlu; pro všechny ostatní komponenty je místo yes nastaveno no.

Pro webový server jsem nastavil *initContainers*, kde čekám v nekonečné smyčce *until*, pokud není dostupná databáze mysql. Udělal jsem to proto, že s každým restartem podu se prodlužuje interval restartů, který chci obejít; a jakmile je databáze dostupná, mohu okamžitě vytvořit webový server.

Jak je patrné z obrázků 5.13 a 5.14, pody jsou rozděleny do jmenných prostorů přesně tak, jak jsem chtěl. Na obrázku 5.14 je vidět, že webový server se restartoval dvakrát, protože databáze sice již může přijímat dotazy, ale není úplně inicializovaná.

Name	Namespace	Containers	Restarts	Controlled By	Node	QoS	Status
mysql-operator-6766d8579c-69r7p	databases	■	0	ReplicaSet	k3d-dev-agent-0	BestEffort	Running
mysql-cluster-0	databases	■■■■■	0	StatefulSet	k3d-dev-agent-0	Burstable	Running
elasticsearch-es-default-0	databases	■■■	0	StatefulSet	k3d-dev-agent-0	Guaranteed	Running
kibana-sample-kb-5dc8f989cc-2s8vd	databases	■■	0	ReplicaSet	k3d-dev-agent-0	Guaranteed	Running
mysql-cluster-router-78c75c9d8-cnx9t	databases	■	0	ReplicaSet	k3d-dev-agent-0	BestEffort	Running

Obr. 5.13: Jmenný prostor databases

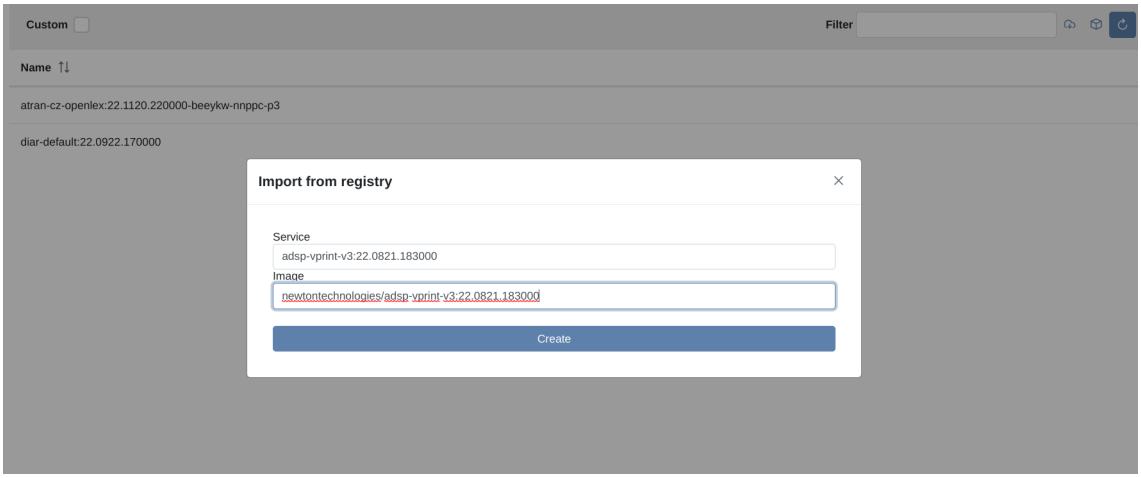
Zdroj: autor

Name	Namespace	Containers	Restarts	Controlled By	Node	QoS	Status
beey-traefik-6b87bf779c-nkxws	default	■	0	ReplicaSet	k3d-dev-agent-1	Burstable	Running
ng-speech-core-7fc995845f-7mrss	default	■	0	ReplicaSet	k3d-dev-agent-1	BestEffort	Running
beey-apps-voice-759b4d4747-84wt8	default	■	0	ReplicaSet	k3d-dev-agent-1	BestEffort	Running
beey-scribe-5458957847-gmbwl	default	■■	2	ReplicaSet	k3d-dev-agent-1	BestEffort	Running
beey-spp-8685fc698f-szst8	default	■	0	ReplicaSet	k3d-dev-agent-1	BestEffort	Running
update-configmap-job-7zhgg	default	■	0	Job	k3d-dev-agent-0	BestEffort	Succeeded

Obr. 5.14: Jmenný prostor default

Zdroj: autor

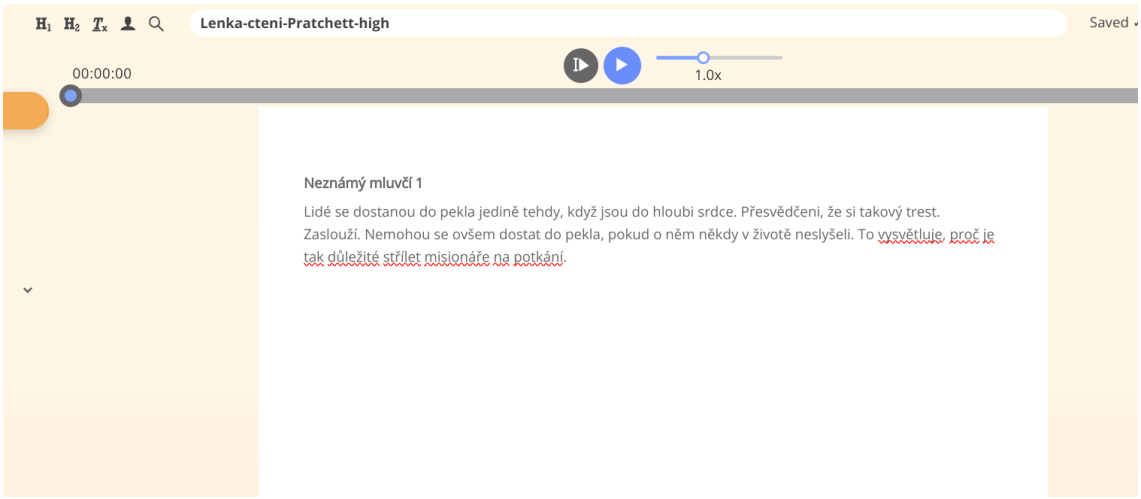
Nyní je třeba nastavit služby nebo modely pro rozpoznávání řeči ve službě jádra. Nevidím velký smysl v použití připojení podle názvu domény, protože testuji lokálně, a proto použiji *port-forwarding*. *Port-forwarding* otevře službu přístupnou na určitém portu a já pak mohu ke službě přistupovat v prohlížeči.



Obr. 5.15: Import modelu do služby jádra

Zdroj: autor

Stejným způsobem se připojím k webovému rozhraní webového serveru a vyzkouším rozpoznávání na zkušebním záznamu. Na obrázku 5.16 je výsledek vývoje, což znamená, že vše funguje, jak má.



Obr. 5.16: Výsledek rozpoznávání řeči na zkušebním záznamu

Zdroj: autor

5.11.2 Prostředí veřejného cloudu

Pro nasazení ve veřejném cloudu je třeba nasadit samotnou infrastrukturu. To se provádí pomocí Terraformu přesně tak, jak jsem nakreslil na obrázku 5.12.

Vytvoření skupiny prostředků a úložiště pro soubor tfstate

```
1  $ source scripts/remote_state.sh
```

Terraformu je třeba dát návod, kde se soubor *tfstate* nachází. Vše jsem již připravil, stačí spustit následující příkaz:

```
$ terraform init -backend-config='backend.cfg'
```

Rozdělil jsem konfigurace pro vývoj a produkci pomocí pracovního prostoru, ale je třeba jej vytvořit.

```
$ terraform workspace new dev
```

Dále potřebuji importovat skupinu prostředků.

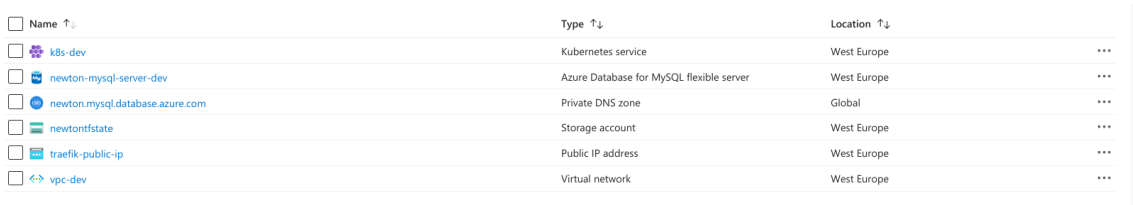
```
$ terraform import azurerm_resource_group.newton-rg /
  subscriptions/41b32a3b-23d9-499d-a7f5-f7c751d2d235/
  resourceGroups/newton-rg
```

Poslední tři příkazy si nemusíte pamatovat, po spuštění prvního skriptu se objeví v konzoli a pak je stačí zkopírovat.

Uplatnění konfigurace bez nutnosti potvrzení

```
$ terraform apply --auto-approve
```

Obrázek 5.17 zobrazuje nasazenou infrastrukturu v rozhraní webového portálu Azure.



Obr. 5.17: Přehled nasazené infrastruktury v Azure

Zdroj: autor

Nasazení do clusteru Kubernetes v Azure se příliš neliší, jedinou drobnou změnou je zadání statické IP adresy pro připojení přes název domény a samozřejmě koupení samotné domény.

Nejprve je však třeba zadat kontext clusteru Kubernetes v Azure do *kubectl*, což jsem také připravil, stačí spustit následující příkaz:

```
$ terraform output -raw kube_config > ~/.kube/config
```

Bohužel příkaz *terraform destroy* nestačí k odstranění infrastruktury, protože jsem zakázal mazání databáze. Provést to může pouze správce prostřednictvím webového portálu, ale stačí odstranit pouze skupinu prostředků.

5.12 Porovnání nasazení v privátním a veřejném cloudu

V této části porovnám nasazení v privátním a veřejném clusteru.

Zásadní rozdíl mezi nimi je v tom, že u veřejného cloudu mohu automatizovat potřebnou infrastrukturu včetně nasazení samotného clusteru Kubernetes. Mohl jsem jít ještě dál a přidat nasazení Terraformu s Helmem dohromady, ale rozhodl jsem se to neudělat, protože považuji za vhodnější rozdělit proces do několika fází.

Jedním z klíčových rozdílů je výběr hardwaru, protože tento faktor je nedílnou součástí celkové ceny celé infrastruktury ve veřejném cloudu. V privátním cloudu vypadá proces instalace samozřejmě mnohem jednodušeji, nehledě na to, že je třeba nasadit cluster Kubernetes. Nesnažím se tvrdit, že privátní cloudy v režimu on-premise jsou nejlepší a nejlevnější možností, to v žádném případě. Nesnažím se ale ani tvrdit opak; jen chci poukázat na klíčový faktor – při použití mého řešení se vyplatí přizpůsobit konfiguraci.

Flexibilita je dána tím, že existuje možnost volby místa nasazení databází, v čemž se také obě metody liší. Privátní cloud totiž znamená plnou kontrolu nad daty a zdroji, což znamená, že neexistuje možnost plně udržovaných databází třetí stranou. To zase znamená, že pokud nejsou databáze vyvíjeny v clusteru Kubernetes, je třeba je předem připravit.

Závěr

Závěrem lze říci, že v rámci bakalářského projektu se podařilo následující:

- Navrhnout přenosné řešení pro nasazení v privátních i veřejných cloudech.
- Nabídnout alternativu v podobě možnosti volby mezi nasazením datové sady v systému Kubernetes a nenasazením.
- Navrhnout řešení tak, aby konfigurace a použití pro privátní a veřejný cloud byly co nejjednodušší a podobné.

Tato bakalářská práce může sloužit jako základ pro seznámení se s tím, které kroky se vyplatí pro nasazení aplikace v cloudovém prostředí.

Co se týče vylepšení do budoucna, ve veřejném cloudu by mohly být přidány další komponenty pro zabezpečení přístupu k aplikaci z internetu. V Kubernetes by stálo za to přidat monitorování. Předem je ale těžké spočítat, jak moc bude řešení pro monitorování zatěžovat cluster. Vymyslet obecné řešení je obtížné, protože není předem známo, co správce clusteru potřebuje.

Během své bakalářské práce jsem se naučil znát a používat spoustu nových technologií a konceptů souvisejících s distribuovanými systémy. Projekt mi poskytl velké zkušenosti s používáním technologií relevantních pro trh a umožnil mi vžít se do role inženýra, který navrhuje a realizuje vlastní řešení.

Seznam použité literatury

[1] SHUISKOV, Alexander. *Microservices with Go*. 1. vyd. Packt Publishing Ltd., 2022. ISBN 9781804617007. Dostupné také z: https://subscription.packtpub.com/book/programming/9781804617007/pref/preflvl1sec10/download-a-free-pdf-copy-of-this-book.

[2] *What is virtualization?* IBM. Dostupné také z: https://www.ibm.com/topics/virtualization.

[3] *What is a container?* Docker. Dostupné také z: https://www.docker.com/resources/what-container/.

[4] MIELL, Ian. *Docker in Action*. 2. vyd. Manning Publications, 2019. ISBN 9781617294808.

[5] *Containerd main web page*. containerd Authors. Dostupné také z: https://containerd.io/.

[6] HOHN, Alan. *The Book of Kubernetes*. Random House LCC US, 2019. ISBN 9780134852211.

[7] *Kubernetes documentation*. The Kubernetes Authors. Dostupné také z: https://kubernetes.io/docs/home/.

[8] LUKSA, Marko. *Kubernetes in Action*. Manning Publications, 2017. ISBN 9781617293726.

[9] MARINESCU, Dan C. *Cloud Computing*. Morgan Kaufmann Publishers, 2022. ISBN 9780323852777.

[10] *To run or not to run a database on Kubernetes: What to consider*. Google. Dostupné také z: https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider.

[11] *Terraform home web page*. Hashicorp. Dostupné také z: https://www.terraform.io/.

[12] *InnoDB Cluster documentation*. Oracle Corporation. Dostupné také z: https://dev.mysql.com/doc/refman/8.0/en/mysql-innodb-cluster-introduction.html.

[13] *MySQL operator documentation*. Oracle Corporation. Dostupné také z: https://dev.mysql.com/doc/mysql-operator/en/mysql-operator-properties.html.

[14] *Elastic on Cloud API Reference*. Elasticsearch. Dostupné také z: https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-api-reference.html.

[15] *The Chart Repository Guide*. Helm Authors. Dostupné také z: https://helm.sh/docs/topics/chart_repository/.

[16] *Cloud Spending Growth Rate Slows But Q4 Still Up By $10 Billion from 2021; Microsoft Gains Market Share*. Synergy Research Group, 2023. Dostupné také z: https://www.srgresearch.com/articles/cloud-spending-growth-rate-slows-but-q4-still-up-by-10-billion-from-2021-microsoft-gains-market-share.

[17] *What is K3s?* K3s Project. Dostupné také z: https://docs.k3s.io/.