Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky



Technické aspekty provozu chatbota

DIPLOMOVÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Informační systémy a technologie

Autor: Bc. Jan Tlustý

Vedoucí diplomové práce: Ing. Filip Vencovský, Ph.D.

Praha, duben 2019

Prohlášení

Prohlašuji, že jsem diplomovou práci Technické aspekty provozu chatbota vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 28. dubna 2019 .....

Jan Tlustý

Poděkování

Děkuji vedoucímu práce Ing. Filipu Vencovskému, Ph.D. za odborné vedení a připomínky.

Abstrakt

Tato práce se zabývá problematikou technických aspektů provozu chatbotů. Jejím cílem je tvorba vlastního prototypu chatbota za použití výhradně open source softwarových nástrojů. Dílčím cílem pak je porovnání open source frameworků pro tvorbu chatbotů.

V práci je nejprve představeno téma chatbotů, je stručně shrnuta jejich historie a jsou uvedeny některé příklady jejich současného využití. Rovněž jsou rozebrány možnosti jejich využití podle rolí uživatelů.

V další části je popisována tvorba prototypu. Nejprve jsou zvoleny odpovídající open source softwarové nástroje a je vytvořeno virtuální prostředí, ve kterém je prototyp dále vytvářen. Tvorba prototypu je podrobně komentována. Po vytvoření prototypu jsou ověřeny jeho technické parametry, které jsou na závěr zhodnoceny.

Klíčová slova

chatbot, škálovatelnost, dostupnost, Docker, Kubernetes.

Abstract

This thesis is focused on technical aspects of running chatbots. Its aim is to create a custom chatbot prototype using solely open source software tools. As a secondary goal, this thesis also aims to compare available open source frameworks for creating chatbots.

Firstly, the topic of chatbots is introduced, a short summary of the chatbot history is presented and several examples of chatbot use cases nowadays are shown. Also use cases by user roles are shown.

In the next part of the thesis, the process of building the prototype is described. After choosing appropriate open source software tools, the virtual environment in which the prototype will be built is set up. The creation of the prototype is commented in detail. When the prototype is complete, its technical parameters are verified and evaluated.

Keywords

chatbot, scalability, availability, Docker, Kubernetes.

Obsah

Úvod 12
1 Rešerše zdrojů 13
2 Co jsou chatboti 14
2.1 Historie chatbotů 14
2.2 Nedávný vývoj 14
2.2.1 Apple Siri 14
2.2.2 IBM Watson 15
2.2.3 Microsoft Cortana 15
2.2.4 Amazon Alexa 15
2.2.5 Facebook Messenger 15
2.2.6 Google Assistant 15
2.2.7 Nástroje pro tvorbu a provoz chatbotů 16
2.3 Možnosti využití chatbotů 17
2.3.1 Podle rolí 17
2.3.2 Podle odvětví 18
2.4 Způsoby fungování chatbotů 20
2.4.1 Chatboti založeni na pravidlech 20
2.4.2 NLP a NLU 21
3 Technické aspekty provozu chatbota 23
3.1 Dostupnost 23
3.1.1 Definice 23
3.1.2 Dostupnost chatbotů 23
3.2 Škálovatelnost 24
3.2.1 Vertikální škálování 24
3.2.2 Horizontální škálování 24
3.2.3 Škálovatelnost chatbotů 24
4 Prototyp chatbota 26
4.1 Zkoumané frameworky pro tvorbu chatbota 26
4.1.1 Botkit 26
4.1.2 Botpress 28
4.1.3 Rasa Stack 29
4.2 Výběr frameworku 30
4.2.1 Kritéria pro výběr 30

4.3 Kontejner 31
4.4 Kubernetes 32
4.5 Databáze 33
5 Tvorba Prototypu 35
5.1 Popis prostředí 35
5.2 Vytvoření virtuálních serverů 36
5.3 Databáze 37
5.4 Nasazení Kubernetes 37
5.5 Nasazení chatbota 40
6 Ověření vlastností prototypu 46
6.1 Škálovatelnost 46
6.2 Vysoká dostupnost 51
6.2.1 Ukončení aplikace v kontejneru a ukončení kontejneru 52
6.2.2 Vypnutí serveru 54
7 Zhodnocení výsledků 59
Závěr 61
Naplnění cílů 61
Přínosy práce 62
Možnosti rozšíření práce 62
Použitá literatura 63
Přílohy 67
Příloha A: Skript provison.sh 67
Příloha B: Konfigurační soubor pro Vagrant 69

Seznam obrázků

Obrázek 1 Historie chatbotů [5] 16
Obrázek 2 Uživatelsky přívětivá tvorba chatbota v motion.ai [9] 17
Obrázek 3 Objednání cestovního pojištění pomocí chatbota [10] 19
Obrázek 4 Příklad chatbota založeného na pravidlech [zdroj: autor] 20
Obrázek 5 Příklad architektury AI chatbota [12] 21
Obrázek 6 Vztah NLP, NLU a NLG [15] 22
Obrázek 7 Vertikální škálování [zdroj: autor] 24
Obrázek 8 Horizontální škálování [zdroj: autor] 24
Obrázek 9 Botkit [23] 28
Obrázek 10 Botpress [25] 29
Obrázek 11 Rasa Stack [zdroj: autor] 30
Obrázek 12 Architektura prototypu [zdroj: autor] 35
Obrázek 13 Botpress – vytvoření účtu [zdroj: autor] 41
Obrázek 14 Botpress – vytvoření chatbota [zdroj: autor] 41
Obrázek 15 Botpress – webový chat [zdroj: autor] 47
Obrázek 16 Firefox – nastavení SOCKS proxy [zdroj: autor] 49
Obrázek 17 Botpress – webový chat přes SOCKS proxy [zdroj: autor] 50

Seznam tabulek

Tabulka 1 Porovnání frameworků [zdroj: autor]....31
Tabulka 2 Mapování portů [zdroj: autor] 36
Tabulka 3 Test obnovy po ukončení kontejneru [zdroj: autor]....54
Tabulka 4 Test obnovy po vypnutí serveru [zdroj: autor]....57

Seznam výpisů programového kódu

Výpis 1 IP adresy serveru kube01[zdroj: autor] 37
Výpis 2 Inicializace Kubernetes clusteru [zdroj: autor] 38
Výpis 3 Připojení serveru kube01 do Kubernetes clusteru [zdroj: autor] 39
Výpis 4 Seznam serverů v Kubernetes clusteru [zdroj: autor] 40
Výpis 5 Spuštění Docker kontejneru [zdroj: autor] 41
Výpis 6 Obsah PostgreSQL databáze [zdroj: autor] 42
Výpis 7 Dockerfile [zdroj: autor] 43
Výpis 8 Sestavení upraveného Docker kontejneru [zdroj: autor] 43
Výpis 9 Přihlášení na Docker Hub [zdroj: autor] 44
Výpis 10 Nahrání upraveného kontejneru na Docker Hub [zdroj: autor] 44
Výpis 11 Vytvoření Kubernetes deploymentu [zdroj: autor] 44
Výpis 12 Jeden spuštěný kontejner [zdroj: autor] 45
Výpis 13 Dva spuštěné kontejnery [zdroj: autor] 46
Výpis 14 Spuštěné kontejnery na serverech [zdroj: autor] 46
Výpis 15 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube01 [zdroj: autor] 47
Výpis 16 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube02 [zdroj: autor] 47
Výpis 17 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube01 – první test [zdroj: autor] 48
Výpis 18 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube02 – první test [zdroj: autor] 48
Výpis 19 Neúspěšný DNS dotaz na doménové jméno „master“ [zdroj: autor] 49
Výpis 20 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube01 – druhý test [zdroj: autor] 50
Výpis 21 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube02 – druhý test [zdroj: autor] 51
Výpis 22 Test obnovy po ukončení kontejneru [zdroj: autor] 53
Výpis 23 Test obnovy po vypnutí serveru [zdroj: autor] 56

Seznam zkratek

AI	Artificial intelligence
DNS	Domain Name System
NFS	Network File System
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NPM	Node Package Manager
PID	Process Identification Number
SSH	Secure Shell

Úvod

Tato diplomová práce se zabývá technickými aspekty provozu chatbotů. Cílem práce je tvorba vlastního prototypu chatbota za použití open source softwarových nástrojů. Dílčím cílem pak je porovnání tří nejoblíbenějších open source frameworků pro tvorbu chatbotů.

V teoretické části práce je nejprve zpracována rešerše dané problematiky, poté jsou vysvětleny pojmy, se kterými se v práci dále pracuje, a popsány některé základní přístupy k tvorbě chatbotů.

V praktické části jsou nejprve definována kritéria pro porovnání chatbotů a následně jsou stručně popsány v současnosti nejpoužívanější open source frameworky pro tvorbu chatbotů. Tyto jsou dále porovnány podle dříve stanovených kritérií. Z porovnání vzejde framework s nejširšími možnostmi pro tvorbu chatbotů a ten je použit pro sestavení prototypu chatbota, který by měl mít co nejlepší technické parametry zmíněné v teoretické části.

Postup sestavení tohoto prototypu je detailně popsán a jsou diskutována jednotlivá rozhodnutí, která byla během práce na něm učiněna. Sestavení prototypu mimo jiné zahrnuje řešení škálovatelnosti a vysoké dostupnosti chatbota a dalších technických aspektů zmíněných v předchozích částech práce.

K dosažení tohoto cíle jsou použity další open source nástroje, které nejsou určeny výlučně pro fungování společně s chatboty, ale jsou pro tento způsob využití vhodné. Práce zjišťuje, zda je možné sestavit a provozovat chatbota postaveného pouze na open source nástrojích, který splňuje všechny požadavky, které mohou být na chatboty v současnosti kladeny.

Téma chatbotů je v současnosti vysoce aktuální, protože velká část komunikace mezi lidmi se přesouvá na sociální sítě a chatboti jsou způsob, jakým lze nabídnout uživatelům různých služeb možnost tyto služby ovládat právě z prostředí sociálních sítí a instant messaging aplikací, které se v současnosti těší velké oblibě a přes které lidé přirozeně komunikují.

1 Rešerše zdrojů

Klíčovou otázkou, kterou si je nutné při provozu libovolné informatické služby položit, je, na jakém prostředí a na jaké platformě bude provozována. Z dostupných zdrojů lze dohledat, že v případě provozu chatbota nejsou vhodné monolitické architektury a vhodnějšími přístupy jsou ty, které výslednou službu skládají z dílčích komponent, tzv. mikroslužeb.

Tímto přístupem se zabývá článek s názvem *Building a Chatbot with Serverless Computing* [1]. Autoři popisují zejména možnosti využití serverless architektury pro chatboty. Svůj prototyp chatbota staví na serverless platformě Apache OpenWhisk. Technické aspekty jako škálovatelnost a vysokou dostupnost řeší skládáním funkčnosti chatbota z jednotlivých mikroslužeb provozovaných v rámci Apache OpenWhisku. Ten je schopen na základě definovaných pravidel jednotlivé části služby škálovat podle jejich aktuálního využití. Velkou výhodou použití mikroslužeb je podle autorů to, že jednotlivé funkcionality jsou na sobě nezávislé, mohou být založeny na různých technologiích a napsány v různých programovacích jazycích. Autoři poznamenávají, že zatímco jimi zvolená platforma Apache OpenWhisk jim umožňuje provozovat chatbota na vlastním hardware, obdobnou konfiguraci chatbota by bylo možné provozovat i na běžných cloudových serverless platformách (AWS Lambda, Microsoft Azure atd.).

Serverless architekturu v souvislosti s provozem chatbota se zabývá i práce *Pixie: A Social Chatbot* [2]. Ta popisuje prototyp chatbota nazvaného Pixie. Ovšem zde se autoři rozhodli pro cloudovou platformu AWS Lambda provozovanou společností Amazon, zde je dostupnost a škálovatelnost zajištěna poskytovatelem platformy. Pro výpočetně náročnější a déle trvající operace autoři Pixie používají Docker kontejnery a k rozložení zátěže mezi nimi využívají webový server Nginx. Docker kontejnery jsou řízeny a úlohy na nich jsou spouštěny z mikroslužeb na AWS Lambda. Pro ukládání dat a uživatelských relací Pixie využívá databázi DynamoDB. Stejně jako v případě AWS Lambda se jedná o službu poskytovanou společností Amazon a její škálovatelnost a dostupnost je zajištěna na straně poskytovatele.

Článek *Realtime Processing of IoT Events using a Software as a Service (SaaS) Architecture with Graph Database* [3] popisuje další možnosti využití AWS pro provozování chatbota. Podobně jako v minulém případě autoři vyzdvihují možnosti škálování služeb využívajících platformu AWS Lambda. Narozdíl od autorů Pixie ale pro ukládání dat používají databázi MySQL. Ta má sice široké možnosti škálování a zajišťování dostupnosti, ale autoři toto dále nerozvádějí.

Vedle architektury služby je nutné uvažovat i o použitém databázovém systému pro ukládání dat. Tímto tématem se zabývá článek *Programming Tools for Messenger-Based Chatbot System Organization: Implication for Outpatient and Translational Medicines* [4]. Autoři v něm popisují využití NoSQL databáze ElasticSearch pro provoz chatbota. Ta umožňuje snadné horizontální škálování pouhým spuštěním další instance a ve výchozím nastavení jsou data replikována mezi více servery.

2 Co jsou chatboti

Pojem chatbot lze definovat jako počítačový program, který zpracovává vstup ve formě lidského přirozeného jazyku a v závislosti na něm vygeneruje smysluplnou odpověď, kterou odešle zpět uživateli [5, s. 1].

2.1 Historie chatbotů

Za počátek historie chatbotů lze považovat rok 1964, kdy byl vyvinut první chatbot nazvaný Eliza. Vyvinul jej Joseph Weizenbaum na Massachusettském technologickém institutu (MIT). Chatbot vystupoval jako terapeut a převážně ve svých odpovědích pouze přeformuloval vstup od uživatele. Chatbot ale nerozuměl významu toho, co mu uživatel sdělil [6, s. 3].

Dalším významným milníkem byl rok 1991. Tehdy byla poprvé udělena Loebnerova cena. Ta se nyní každoročně uděluje vítězi soutěže programů umělé inteligence. Soutěžící programy se snaží úspěšně projit Turingovým testem. Tedy testem, kdy se člověk snaží rozpoznat, jestli komunikuje s jiným člověkem nebo systémem umělé inteligence [7, s. 17-19]. Během doby trvání soutěže v ní bylo představeno několik zajímavých chatbotů. Mezi nimi jsou např. chatboti ALICE, JabberWacky a Rose. Nicméně až do roku 2014 Turingovým testem žádný z nich úspěšně neprošel. Prvním chatbotem, kterému se to podařilo, byl chatbot nazvaný Eugene Goostman, který měl vystupovat jako třináctileté dítě [6, s. 9].

Významnou událostí pro vývoj chatbotů byl také vznik značkovacího jazyku AIML. Ten umožnil relativně snadné definování chování chatbotů pomocí definování vzorů, se kterými byly následně porovnávány vstupy od uživatele [8, s. 86].

2.2 Nedávný vývoj

Zde následuje výčet významných chatbotů a událostí z nedávné minulosti a současnosti [6, s. 10-12].

2.2.1 Apple Siri

V současné době je pravděpodobně nejznámějším příkladem chatbota osobní asistent Siri od společnosti Apple. Ten byl vydán v roce 2011 a lze jej používat pro ovládání telefonu. Pomocí Siri je možné uskutečňovat hovory, psát a nechat si číst zprávy, vyhledávat na internetu nebo např. ovládat kalendář.

2.2.2 IBM Watson

Rovněž v roce 2011 představila společnost IBM systém IBM Watson. Jedná se o systém, jehož účelem je analyzovat otázky kladené v přirozeném lidském jazyce a v reálném čase na ně odpovídat.

2.2.3 Microsoft Cortana

Podobně jako telefony od společnosti Apple jsou vybaveny virtuálním asistentem Siri, Microsoft přidal v roce 2013 do telefonů s Windows Phone vlastního virtuálního asistenta Cortana. Později se tento virtuální asistent objevil i v počítačích s operačním systémem Windows 10.

2.2.4 Amazon Alexa

V roce 2014 představila společnost Amazon vlastního virtuálního asistenta Alexa. Ten je dostupný v zařízení nazvaném Echo, který společnost Amazon prodává. Echo je první zařízení svého druhu. Jedná se o reproduktor se zabudovanou výpočetní jednotkou. Zařízení reaguje na hlasové příkazy od uživatele a na rozdíl od asistentů v chytrých telefonech a počítačích je hlasová komunikace jedinou možnou cestou komunikace s tímto zařízením.

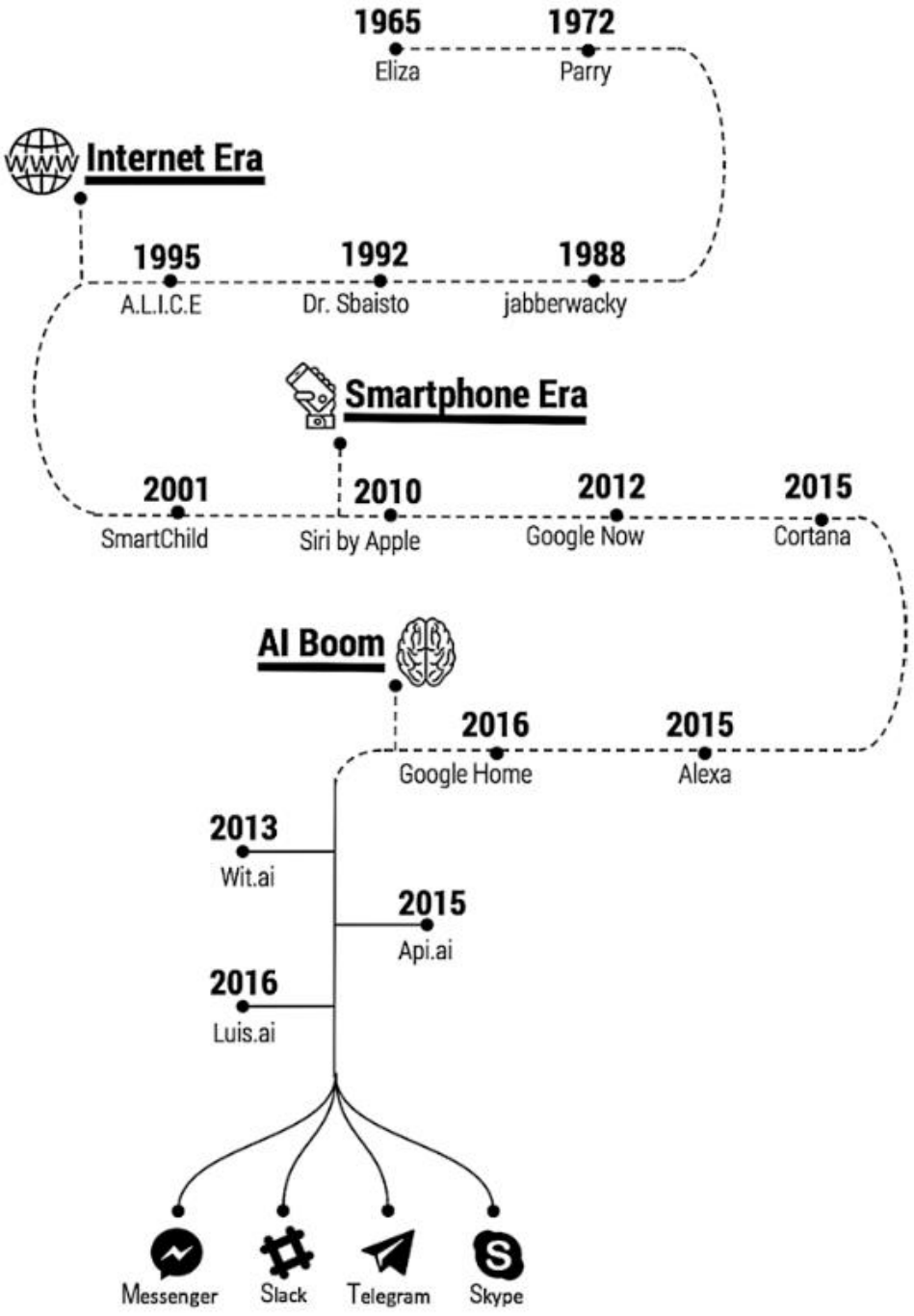
2.2.5 Facebook Messenger

Velmi podstatnou událostí pro svět chatbotů bylo, když v roce 2016 společnost Facebook ohlásila, že umožní napojení chatbotů na svou komunikační platformu Facebook Messenger. Toto otevření vedlo k vzniku velkého počtu nových chatbotů, protože možnosti jejich využití se podstatně rozšířily a bylo s jejich pomocí možné komunikovat s o mnoho větším počtem lidí. V reakci na toto umožnily připojení chatbotů i další komunikační platformy, např. Skype a Telegram.

2.2.6 Google Assistant

V roce 2016 ohlásila společnost Google vlastního virtuálního asistenta nazvaného Assistant. Toho je možné využívat jak z chytrého telefonu, tak i ze zařízení Google Home, což je obdoba zařízení Echo od společnosti Amazon. Assistant byl původně dostupný exkluzivně na telefonech vyrobených společností Google (řada Pixel). Od roku 2017 je dostupný i na ostatních telefonech s operačním systémem Android.

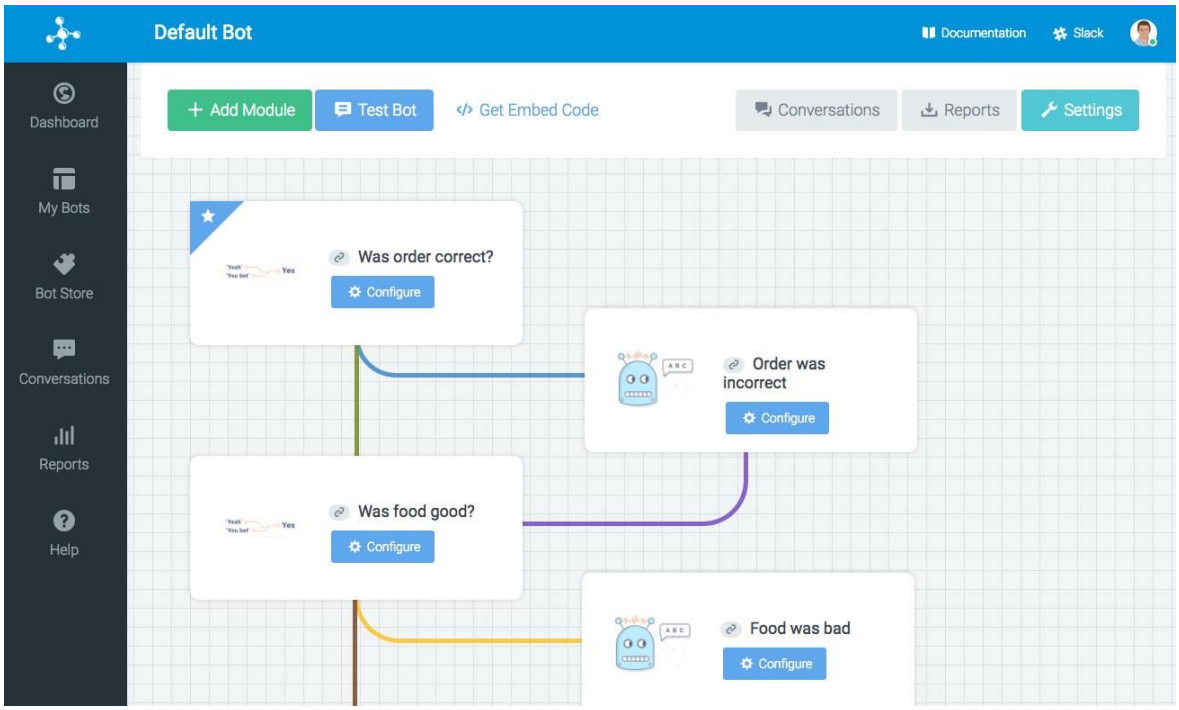
Historický vývoj chatbotů povedenou formou ilustruje Obrázek 1.



Obrázek 1 Historie chatbotů [5]

2.2.7 Nástroje pro tvorbu a provoz chatbotů

Spolu s vývojem chatbotů v chytrých telefonech a spotřební elektronice docházelo i k rychlému vývoji na poli nástrojů pro tvorbu a provoz chatbotů. Tvorba chatbotů se za poslední roky díky novým uživatelsky přívětivým nástrojům (viz Obrázek 2) podstatně zjednodušila.



Obrázek 2 Uživatelsky přívětivá tvorba chatbota v motion.ai [9]

2.3 Možnosti využití chatbotů

Možnosti využití chatbotů lze rozdělovat podle rolí, v nichž se jejich uživatelé nacházejí, a podle odvětví, kde je konkrétní chatbot využíván [5, s. 98-106].

2.3.1 Podle rolí

Business-to-Business (B2B)

Při interakci mezi podniky mohou chatboti zastávat funkci asistentů, kteří například objednávají produkty a služby, které daný podnik potřebuje pro svoje bezproblémové fungování. Toto využití chatbotů se nabízí, protože tyto činnosti lze poměrně snadno zautomatizovat.

Business-to-Consumer (B2C)

Ve většině případů chatboti přímo nabízejí produkty nebo služby. Chatbot také může poskytovat informace o podniku nebo o jím nabízených produktech a službách. Případně lze chatbota využít pro činnosti jako přijímání reklamací nebo připomínek ke službám podniku.

Business-to-Employee (B2E)

Za poslední roky vzniklo mnoho nových kanálů, po kterých může podnik komunikovat se svými zaměstnanci. Velká část této komunikace může být zautomatizována pomocí chatbotů. Chatboty je možné propojit s podnikovým HR systémem a data z něj používat pro komunikaci. Příkladem takových využití může být např. žádost o dovolenou, nahlášení nemoci atd.

Employee-to-Employee (E2E)

S rozvojem komunikačních platforem jako Slack nebo Skype for Business došlo i k přesunu velké části komunikace mezi zaměstnanci právě na tyto komunikační platformy. Do komunikace mezi zaměstnanci lze na těchto platformách snadno zapojit chatboty. Ti mohou v rámci těchto konverzací např. vyhledávat informace, které si zaměstnanci vyžádají, nebo uživatele upozorňovat na určité události.

2.3.2 Podle odvětví

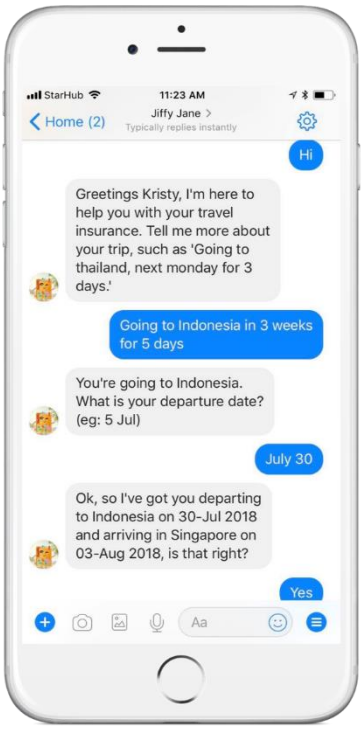
V následujících odvětvích je značný potenciál pro využití chatbotů [5, s. 98-106].

Bankovnictví

Velké množství v současnosti nabízených bankovních služeb může být přístupné přes chatovací rozraní. To mj. zahrnuje nalezení nejbližší pobočky, bankomatu, dotaz na zůstatek na účtu atd. Chatbot také zvládne vyřídit požadavky jako zablokovaní ztracené platební karty, žádost o vydání nové platební karty nebo žádost o kreditní kartu. Zde je velmi důležité, aby byla dobře nastavena oprávnění a mechanismy pro ověření uživatele.

Pojišťovnictví

Oblast pojišťovnictví v sobě obsahuje mnoho výměn dat mezi klientem a pojišťovnou. Velká část těchto dat je strukturovaná a práce s nimi tak může být velmi dobře zautomatizována. Úkony, pro které už jsou v současnosti využíváni chatboti, jsou např. nahlašování pojistné události, kontrola stavu pojistné události a sjednávání pojistných produktů. Na základě znalosti informací o klientech je také možné, aby chatbot aktivně nabízel pojistné produkty na míru. Na Obrázku 3 je zobrazen příklad komunikace s chatbotem v oblasti pojišťovnictví.



Obrázek 3 Objednání cestovního pojištění pomocí chatbota [10]

Cestování

Při cestování lze chatboty používat pro vyhledávání a rezervaci letenek a hotelů. Pro toto sice nyní existuje celá řada webových portálů, ale chatboti jsou schopni nabídnout stejnou funkcionalitu a jejich použití může být pro uživatele intuitivnější. Výhodou je také, že v chatu je uchovávána historie konverzace a uživatel má tak přehled o historii svých vyhledávání a také o zajímavých nabídkách, které mu byly chatbotem na základě jím zadaných kritérií nabídnuty. Další možné využití chatbotů je personalizované vyhledávání zajímavých míst k navštívení.

Pohostinství

V oblasti pohostinství jsou možnosti využití chatbotů velmi široké. Je zde prostor pro různé druhy jednoduchých chatbotů pro přijímání objednávek nebo rezervace stolů. Méně obvyklým, ale existujícím, příkladem jsou chatboti, kteří fungují jako barmani a jsou schopni např. na základě znalosti oblíbených chutí hosta doporučit drink.

E-commerce

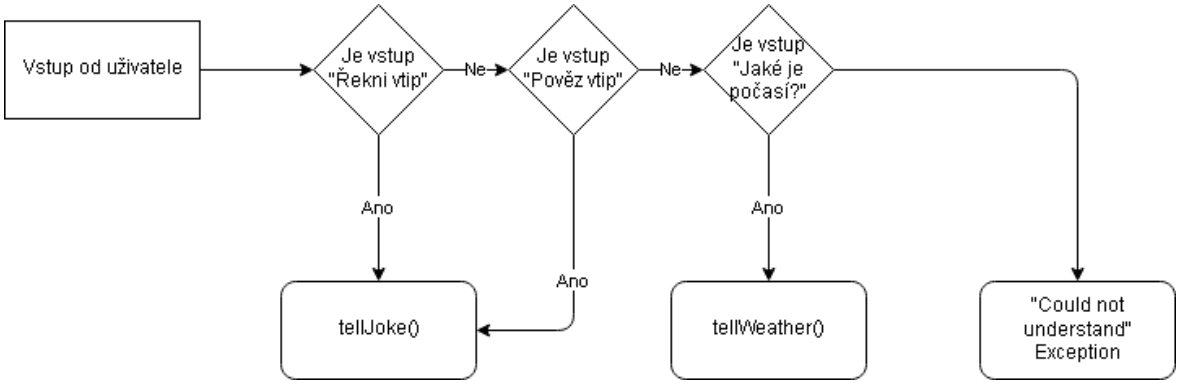
V oblasti e-commerce jsou především dvě hlavní možnosti využití chatbotů. Jednou z nich je vyhledání produktů na základě kritérií zadaných uživatelem a druhou pak zákaznická podpora. Automatizací zákaznické podpory je možné ušetřit velké množství finančních prostředků a zároveň mít jistotu, že zákazníkům se dostává služeb na určité úrovni.

2.4 Způsoby fungování chatbotů

Chatboty lze podle způsobu, jakým zpracovávají vstup od uživatele, rozdělit na následující dvě základní skupiny [5, s. 1].

2.4.1 Chatboti založeni na pravidlech

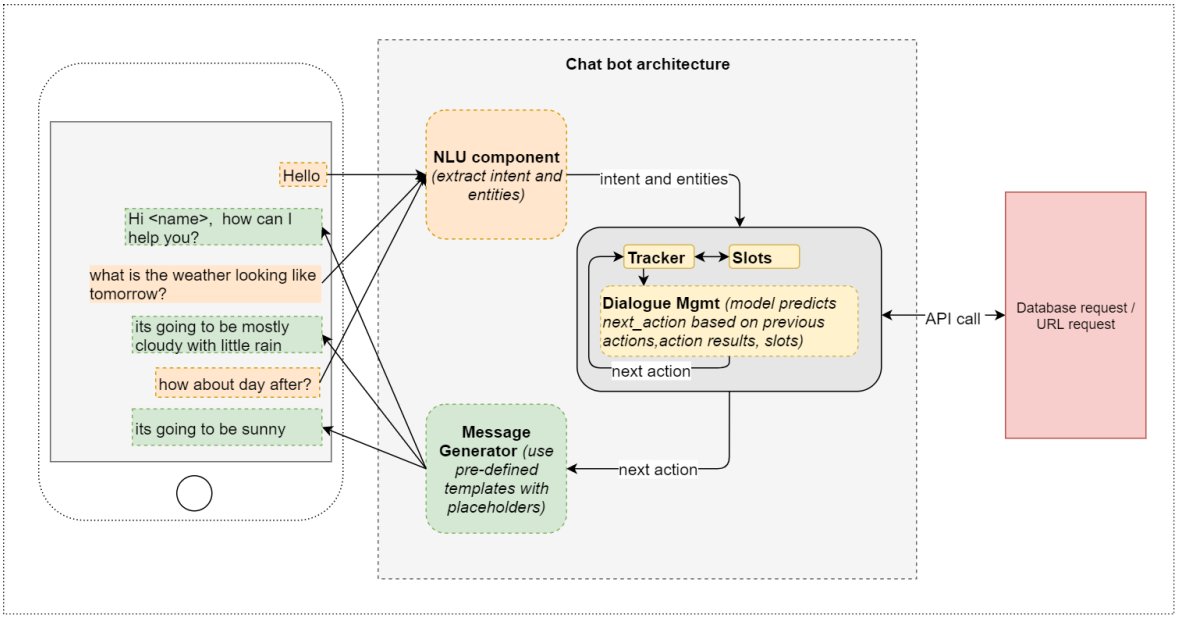
První skupinou jsou chatboti, kteří se při vyhodnocování vstupu on uživatele řídí výhradně předem definovanými pravidly. Tento přístup je velmi přímočarý, protože během vyhodnocování vstupu je pouze v daném pořadí vyhodnoceno, která pravidla vstup splňuje, a podle toho je vygenerována a odeslána odpověď. Nicméně tento přístup není příliš efektivní, zejména z toho důvodu, že stačí ve vstupu malá odchylka od formátu, pro který jsou připravená pravidla, a chatbot nemusí být schopný vstup správně zpracovat. V tomto ohledu je velkou nevýhodou přirozených jazyků, že sdělení v nich mohou mít zpravidla mnoho forem a není snadné se všemi počítat. Příkladem tohoto přístupu může být použití regulárních výrazů, jejichž shoda je testována se vstupem [11, s. 358]. Na Obrázku 4 je zobrazeno schéma velmi jednoduchého chatbota založeného na pravidlech.



Obrázek 4 Příklad chatbota založeného na pravidlech [zdroj: autor]

AI chatboti

Druhá skupina chatbotů využívá pro porozumění vstupu od uživatele umělou inteligenci (AI). Pomocí speciálních algoritmů a strojového učení je usilováno o to, aby chatbot porozuměl významu vstupu od uživatele. To je velký rozdíl oproti chatbotům řídícím se výhradně předem definovanými pravidly, kteří významovou stránku vstupu vůbec neřeší. Tyto chatboty lze trénovat na co možná nejlepší výsledky tím, že jim je dán k dispozici set trénovacích dat obsahujících otázky, odpovědi a například informaci o tom, s jakým úmyslem uživatel každou otázku odeslal [5, s. 27-44]. Příklad schématu AI chatbota je na obrázku 5.

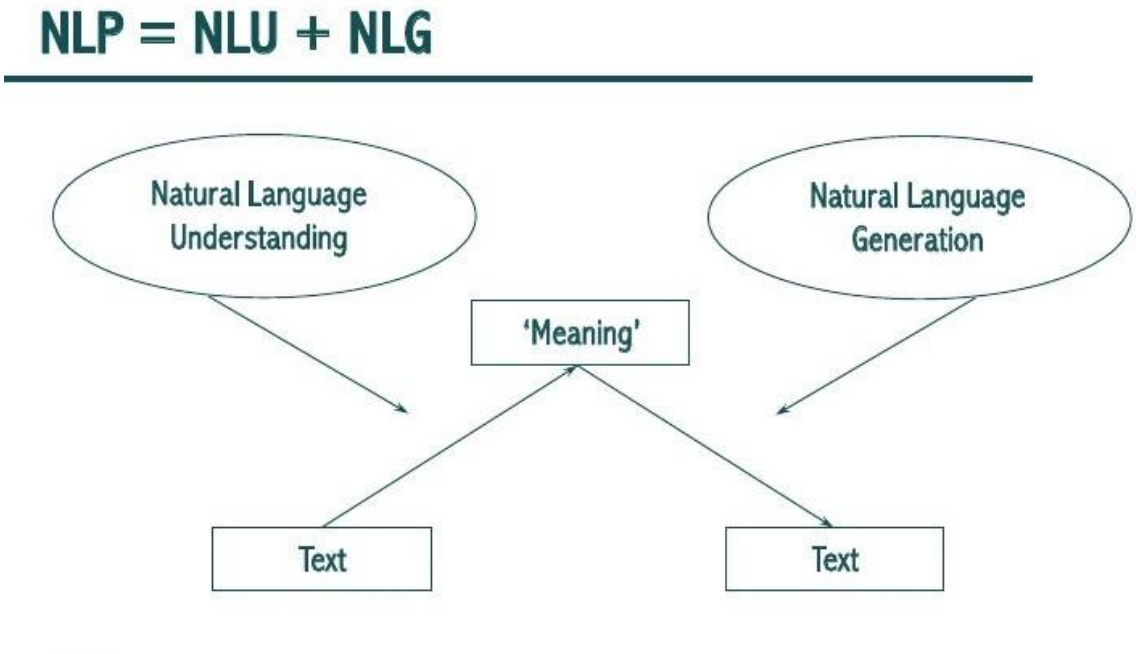


Obrázek 5 Příklad architektury AI chatbota [12]

2.4.2 NLP a NLU

AI chatboti pro svoje fungování využívají tzv. NLU (Natural Language Understanding) a NLP (Natural Language Processing) nástroje. Prvním pojmem rozumíme nástroje, které jsou určeny pro porozumění lidskému přirozenému jazyku. Cílem je, aby počítač rozuměl přirozenému jazyku stejně jako člověk, tedy aby pochopil, co se druhá strana snaží říci [13, s. 12-14].

NLP je pak obecnější pojem, kterým rozumíme nástroje pro zpracování přirozeného jazyku. To v sobě zahrnuje zmíněné NLU nástroje a dále nástroje NLG (Natural Language Generation) (viz Obrázek 6). To jsou nástroje, které slouží pro generování výstupu ve formě přirozeného jazyku, tak aby chatbot mohl na pochopený vstup adekvátně zareagovat [14, s. 42].



Obrázek 6 Vztah NLP, NLU a NLG [15]

NLP probíhá v následujících pěti krocích [14, s. 42]:

- **Lexikální analýza –** Zde je vstup rozdělen na lexikální jednotky (např. čísla, klíčová slova, operátory, věty).
- **Syntaktická analýza –** Řeší rozpoznávání struktury vět, vztahů mezi entitami a funkci jednotlivých slov a frází. Příkladem mohou být věty
    - "Doprava v Praze je špatná."
    - "Šli jsme doprava".

  Slovo "doprava" má zjevně v obou větách zcela jiný význam. Úkolem syntaktické analýzy je zde rozpoznat, že v první větě se jedná a podmět, zatímco v druhé větě se jedná o příslovečné určení místa.

- **Sémantická analýza –** Zkoumá vstup z hlediska jeho významu. Cílem zde je porozumění tomu, co bylo textem myšleno.
- **Zasazení do širšího kontextu –** Při vyhodnocování významu každé věty na vstupu je nutné brát v potaz větu předcházející, která její význam přímo ovlivňuje.
- **Pragmatická analýza –** V tomto posledním kroku se zkoumá význam vstupu vzhledem ke kontextu, v kterém vznikl. Vyžaduje pochopení záměrů a cílů autora vstupu. Počítačem by měly být zjištěny i informace, které v textu nejsou přímo uvedeny.

3 Technické aspekty provozu chatbota

V této kapitole budou představeny technické aspekty, které budou v dalších částech práce u zkoumaných chatbotů sledovány. Dostupné zdroje (zejména případové studie) zabývající se tímto tématem kladou shodně velký důraz na škálovatelnost chatbota a dále zmiňují požadavky na jeho dostupnost [2, 16]. Toto budou dvě hlavní sledované vlastnosti.

3.1 Dostupnost

3.1.1 Definice

Dostupnost systému je ukazatel, který vyjadřuje, jaké množství času sledovaný systém fungoval dle očekávání. Při jejím vyjadřování se pracuje s hodnotami MTBF a MTRR [17, s. 17].

- **MTBF** (Mean Time Between Failures) - Střední doba mezi poruchami
  – Očekávaný čas, po kterém dojde k nefunkčnosti komponenty, pro kterou je hodnota vyjádřena.

- **MTRR** (Mean Time to Repair) - Střední doba opravy
  – Očekávaný čas, potřebný k opravě komponenty, pro kterou je hodnota vyjádřena.

Dostupnost pak lze vyjádřit následujícím vztahem [17, s. 17]:

$$\frac{MTBF}{MTBF + MTTR}$$

Jedná se tedy o poměr střední doby mezi poruchami a součtu střední doby mezi poruchami a střední doby opravy. Za zdůraznění zde stojí fakt, že se jedná o střední hodnoty, statistické veličiny, a neříkají nám, že u popisované komponenty je zaručeno, že k nefunkčnosti nedojde dříve.

3.1.2 Dostupnost chatbotů

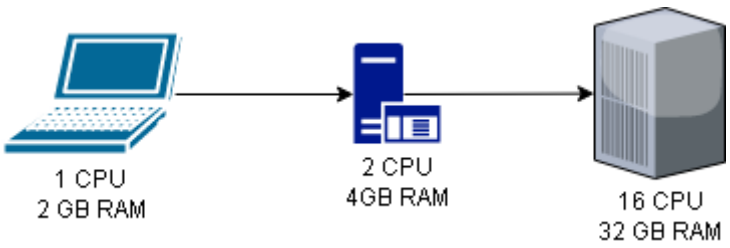
Pro téma chatbotů je potřeba se zabývat dostupností samotné aplikace chatbota, která vykonává funkce chatbota, databázového systému, na který jsou ukládána data, a volitelně dalších komponent, které chatbot využívá, takovou komponentou může být např. samostatný NLP engine.

3.2 Škálovatelnost

Škálovatelnost je schopnost systému fungovat pod zvýšenou zátěží, toho je nejčastěji dosaženo přidáním systémových prostředků [18]. Škálovat lze buď vertikálně nebo horizontálně [19, s. 166].

3.2.1 Vertikální škálování

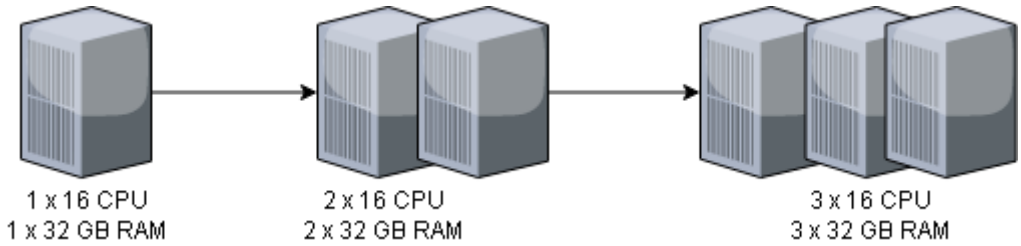
Vertikální škálování znamená zvyšování výkonu již existujících výpočetních jednotek. Typicky jsou do již existujících serverů přidávány komponenty s cílem zvýšit výkon celého systému. Nejčastěji jsou přidávány procesory, operační paměť a další hardware (viz Obrázek 7) [19].



Obrázek 7 Vertikální škálování [zdroj: autor]

3.2.2 Horizontální škálování

Horizontální škálování znamená zvyšování počtu výpočetních jednotek, které v systému vykonávají určitou činnost. Přidáním další výpočetní jednotky se sníží zatížení zbývajících jednotek a systém může pracovat rychleji. Příkladem mohou být webové servery umístěné za load balancerem. Klienti k webovým serverům přistupují přes load balancer, který je nakonfigurován tak, aby jejich požadavky přesměroval vždy na webový server s nejnižším zatížením. Pokud k dostupným webovým serverům přidáme další, na který bude moci load balancer směrovat klienty, bude se jednat o horizontální škálování (viz Obrázek 8) [19].



Obrázek 8 Horizontální škálování [zdroj: autor]

3.2.3 Škálovatelnost chatbotů

Chatboty lze škálovat vertikálně i horizontálně. Vertikální škálování spočívá v prostém přidání hardwarových komponent do systému, na kterém je chatbot spuštěn. Vertikálně lze chatboty škálovat spuštěním více instancí chatbota a rozložením zátěže mezi ně.

Avšak vzhledem k serverless architektuře, která je, jak bylo řečeno, často použita pro chatboty, jsou chatboti obvykle škálováni horizontálně [1]. Je výhodné mít možnost měnit počet instancí aplikace (což v mnoha případech znamená počet kontejnerů) podle aktuálního zatížení, protože tak lze zamezit tomu, aby bylo plýtváno prostředky na provoz předimenzované služby [20, s. 10].

4 Prototyp chatbota

Součástí práce je tvorba prototypu chatbota založeného na některém z dostupných open source frameworků pro tvorbu chatbotů, který bude splňovat požadavky na vysokou dostupnost a bude snadno škálovatelný. Toto je možné rozdělit na dílčí úkoly.

Zaprvé je nutné vybrat vhodný samotný framework pro tvorbu chatbota. Framework musí splňovat několik podmínek. Je vhodné, aby pro ukládání dat používal takový databázový systém, který svoji škálovatelnost a vysokou dostupnost řeší sám, a tedy není nutné tyto aspekty provozu řešit samostatně.

Dále je potřeba řešit dostupnost a škálovatelnost samotného chatbota. Pro toto existují dvě hlavní možnosti [21, s. 7-8].

První možností je, že tyto vlastnosti jsou řešeny na aplikační úrovni chatbota, tj. že aplikace chatbota podporuje provoz v clusteru více serverů, tak aby se v případě výpadku jednoho nebo více z nich služba automaticky přesunula na zbývající servery. Podobně je možné, aby aplikace řešila svou škálovatelnost a v případě potřeby spustila nové instance.

Druhou možností je, že aplikace tyto možnosti nepodporuje a její škálovatelnosti a vysoké dostupnosti je nutné docílit samostatně pomocí různých podpůrných nástrojů.

Bude tedy hledáno takové technické řešení, které umožní provozovat několik instancí určité aplikace chatbota a regulovat jejich počet. Zároveň by takové řešení mělo být odolné proti výpadku některé z instancí a v případě potřeby nefunkční instanci vyřadit a místo ní spustit novou.

Dále musí být zajištěno, aby byly jednotlivé instance propojeny, měly v každý okamžik přístup ke stejným datům a byly přístupné přes jednotné rozhraní (adresu).

4.1 Zkoumané frameworky pro tvorbu chatbota

Pro účely této práce byly vybrány tři nejpopulárnější open source frameworky pro tvorbu chatbotů. Řazeny jsou podle oblíbenosti (počtu hvězd) na GitHubu. V této kapitole budou představeny a budou shrnuty jejich parametry.

4.1.1 Botkit

Botkit je nejpopulárnější open source framework pro tvorbu chatbotů na GitHubu. Je k dispozici pod licencí MIT. Botkit obsahuje podporu pro následující komunikační platformy:

- Slack
- Cisco Webex
- Cisco Jabber
- MS Bot Framework

- MS Teams
- Facebook Messenger
- Twilio
- Google Hangouts

Je napsaný v jazyce JavaScript a mimo zmíněné komunikační platformy umožňuje komunikaci i přes vestavěný webový server. Celý framework je z velké části modulární, je možné ho rozšiřovat pomocí pluginů a je distribuován jako NPM balíček.

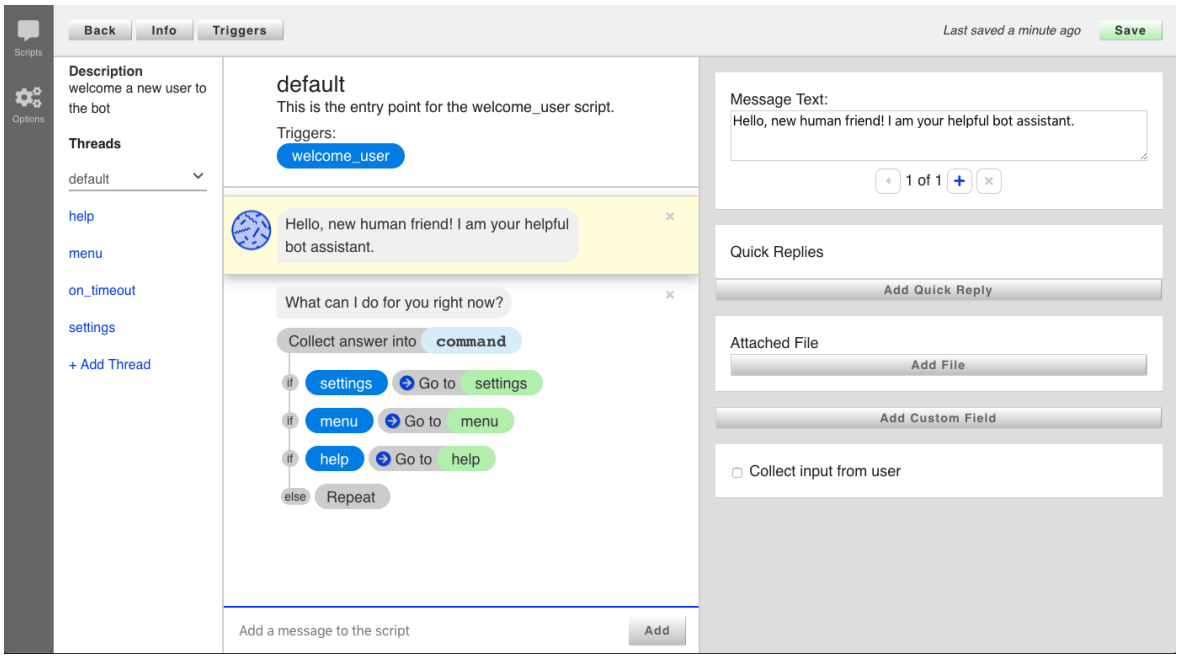
Framework rozlišuje tři druhy pluginů. První kategorií jsou pluginy, které umožňují využívat NLP funkcionality třetích stran. Mezi nimi jsou například Microsoft Luis, Amazon Lex a Google Dialogflow. Botkit nemá vlastní NLP engine.

Druhou kategorií jsou pluginy pro úložiště dat. Ve výchozím nastavení Botkit ukládá veškerá data ve formátu JSON na lokální úložiště. To je sice velmi jednoduché a uživatel se nemusí starat o konfiguraci úložiště, ale pro komplexnější konfigurace může být vhodné použít některý z běžně používaných databázových systémů. Toto umožňují pluginy pro úložiště dat. V tomto ohledu poskytuje Botkit největší možnost volby ze všech porovnávaných frameworků, protože pro něj dostupné pluginy umožňují využívat velké množství databázových systémů. Jsou to:

- MongoDB
- Redis
- Google Cloud Datastore
- Firestore
- Firebase
- Postgres
- CouchDB
- MySQL

Třetí kategorií jsou pluginy zabývající se metrikami, statistikami a CRM. Ty umožňují analyzovat průběh konverzací s uživateli a sledovat metriky definované provozovatelem chatbota. Příkladem těchto metrik může být procento vracejících se zákazníků nebo počet zpráv v konverzaci před vyřešením zákazníkova požadavku.

Pro vývoj chatbota na tomto frameworku jsou dostupné nástroje, které zahrnují tzv. začátečnické sady a také vlastní SDK [22]. Ukázka práce s frameworkem Botkit je na Obrázku 9.



Obrázek 9 Botkit [23]

4.1.2 Botpress

Dalším populárním frameworkem pro tvorbu chatbotů je framework Botpress. Je dostupný pod licencí AGPLv3 a napsaný v TypeScriptu. Chatboty v něm lze vytvářet velmi snadno, programátoři, kteří stojí za jeho vývojem, říkají, že Botpress je pro tvorbu chatbotů to samé, co je Wordpress pro tvorbu webů.

Oproti dříve zmíněnému frameworku Botkit obsahuje Botpress vlastní NLP engine. Navíc jsou podporovány NLP enginy DialogFlow, Microsoft Luis, Recast a Rasa. Autoři Botpressu doporučují využívat jejich vestavěný NLP engine, protože ho považují za bezpečnější (data jsou zpracována lokálně a neopustí server jako v případě využití některé ze služeb třetích stran). Také je oproti nim předvídatelnější, protože nad ním má uživatel plnou kontrolu a nehrozí, že budou aplikovány změny nebo aktualizace, které si uživatel nepřeje.

Pro tvorbu dialogů umožňuje Botpress využít vlastní jednoduchý vizuální editor a rovněž má k dispozici vlastní SDK. Výhodou oproti Botkitu může být i to, že Botpress je distribuován jako samostatná aplikace bez závislostí na externích knihovnách.

Botpress vedle open source verze nabízí i placenou verzi, která navíc nabízí možnost škálování na více serverech na úrovni aplikace, řešení vysoké dostupnosti a rozšířené možnosti správy oprávnění pro uživatele.

Podporovanými komunikačními kanály jsou následující:

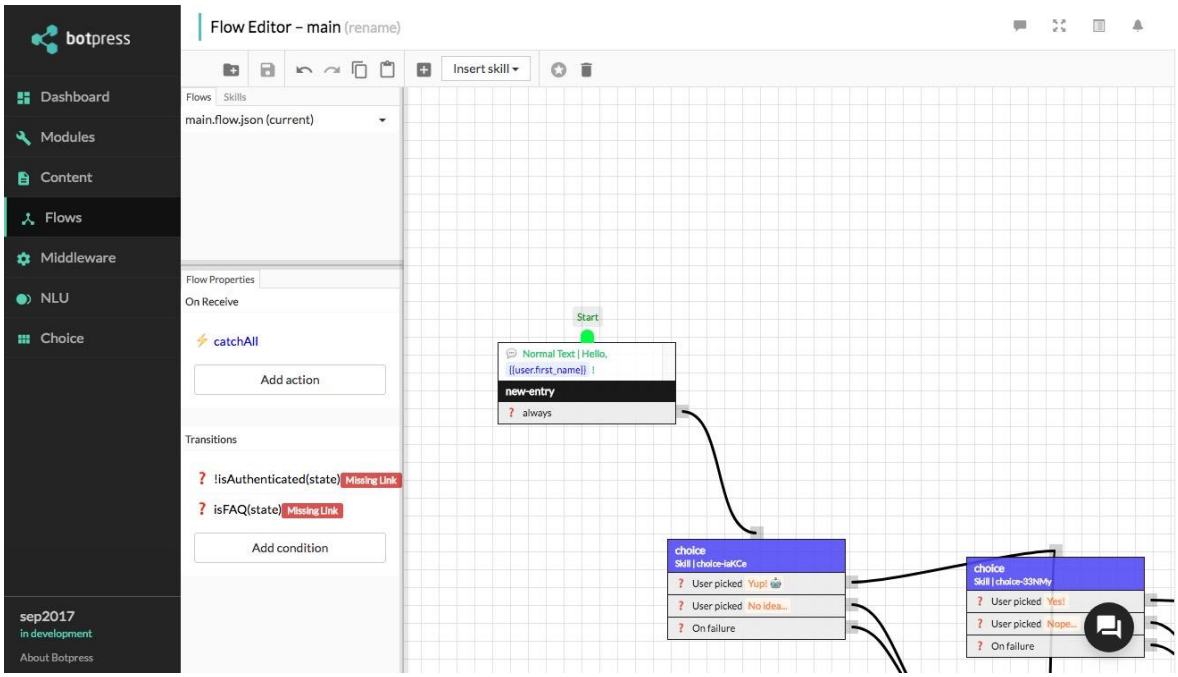
- Facebook Messenger
- MS Bot Framework
- Slack
- Telegram

- Twilio

Navíc Botpress umožňuje komunikaci přes vlastní webové chatovací rozhraní.

Ve výchozím nastavení používá Botpress pro ukládání dat formát SQLite, což není nic jiného než jeden soubor na souborovém systému. Volitelně je možné používat databázové systémy PostgreSQL, MySQL a MongoDB. Vývojáři Botpressu doporučují pro produkční nasazení používat PostgreSQL.

Pro nasazení a provoz chatbotů jsou podporovány cloudové platformy AWS a Heroku. Pro obě vývojáři poskytují podrobný návod včetně propojení s PostgreSQL [24]. Ukázka práce s frameworkem Botpress je na Obrázku 10.



Obrázek 10 Botpress [25]

4.1.3 Rasa Stack

Framework Rasa Stack je napsaný v jazyce Python a šířený pod licencí Apache License 2.0. Z komunikačních kanálů podporuje následující:

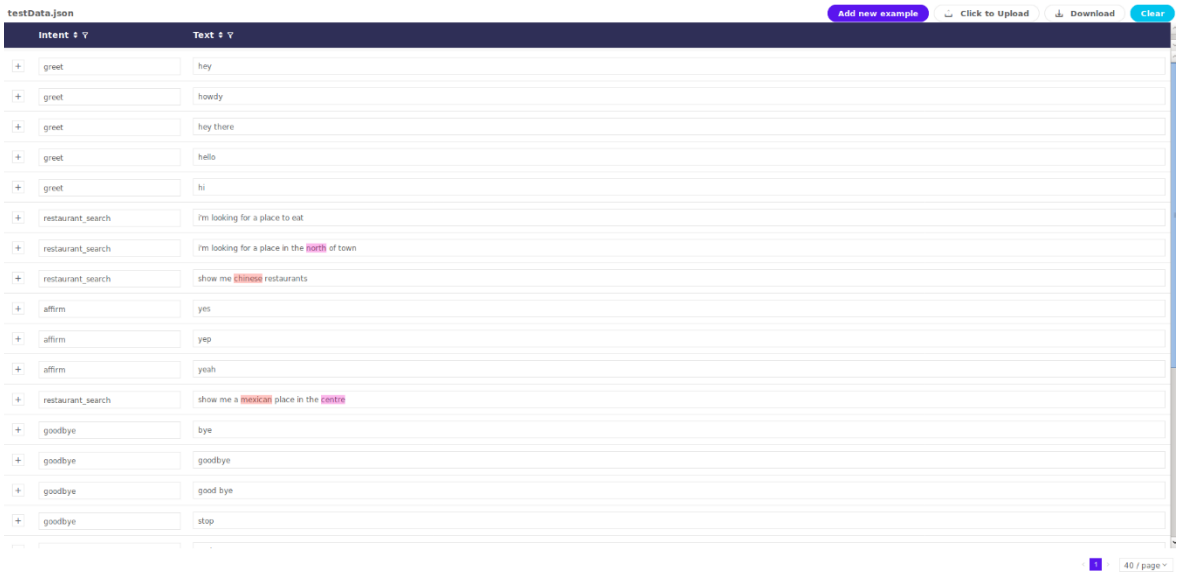
- Facebook Messenger
- Slack
- Telegram
- Mattermost
- Rocket.Chat
- Twilio

Obsahuje vlastní NLP engine a jsou k dispozici manuály pro migraci aplikací z NLP enginů vybraných třetích stran (Google Dialogflow, Microsoft LUIS, Amazon Lex, ...).

Podobně jako autoři frameworku Botpress uvádějí autoři frameworku Rasa Stack, že významným důvodem pro opuštění těchto služeb je, aby data chatbota neopouštěla infrastrukturu jeho provozovatele a provozovatel měl tak jistotu, že s nimi nebude nijak manipulováno bez jeho vědomí.

Vedle frameworku Rasa Stack, který je open source, nabízejí vývojáři i placený balík nazvaný Rasa Platform. Ten k funkcionalitám, které nabízí Rasa Stack, přidává pokročilé API pro konfiguraci a ovládání chatbota a podporu pro škálování na úrovni aplikace.

Rasa Stack ve výchozím nastavení ukládá veškerá data do operační paměti a ta tak nejsou perzistentní. Podporovány jsou databázové systému MongoDB a Redis [26]. Ukázka práce s frameworkem Rasa Stack je na Obrázku 11.



Obrázek 11 Rasa Stack [zdroj: autor]

4.2 Výběr frameworku

V této kapitole bude z frameworků představených výše vybrán ten, který bude nejvhodnější pro zamýšlený prototyp, jehož tvorba je cílem práce.

4.2.1 Kritéria pro výběr

Jak bylo řečeno ve třetí kapitole, velmi často hodnocenými kritérii u chatbotů jsou vysoká dostupnost a škálovatelnost. Další důležitou vlastností je rozšiřitelnost (zde se může jednat o dostupnost rozšíření, které přidají funkcionalitu, nebo možnost naprogramovat vlastní rozšíření) [1]. Dále bude zkoumán počet komunikačních platforem, které jsou frameworkem podporovány, zda framework podporuje NLP enginy třetích stran, zda má framework vlastní NLP engine a zda framework podporuje ukládání dat do některého z open source databázových systémů, které lze provozovat ve vysoce dostupné a škálovatelné konfiguraci. Za výhodu je též považováno, pokud pro framework existuje oficiální kontejner,

ve kterém lze chatboty ve frameworku vytvořené provozovat, což je obecně rozšířený způsob provozu chatbotů [1].

V Tabulce 1 jsou jednotlivé frameworky porovnány podle zmíněných kritérií. Jednotlivé řádky odpovídají jednotlivým kritériím. Pokud za dané kritérium framework získal bod, je text v odpovídající buňce napsán tučným písmem.

Tabulka 1 Porovnání frameworků [zdroj: autor]

	Botkit	**Botpress**	**Rasa Stack**
Nativní škálovatelnost	Ne	Ne	Ne
Nativní vysoká dostupnost	Ne	Ne	Ne
Rozšiřitelnost pomocí pluginů	**Ano**	**Ano**	**Ano**
Vlastní NLP engine	Ne	**Ano**	**Ano**
Podpora NLP enginů třetích stran	**Ano**	**Ano**	Ne
Oficiální kontejner	Ne	**Ano (Docker)**	**Ano (Docker)**
Počet podporovaných komunikačních platforem třetích stran	**8**	5	6
Podpora open source databázových systémů	**Ano** - MongoDB - Redis - PostgreSQL - CouchDB - MySQL	**Ano** - PostgreSQL - MySQL - MongoDB	**Ano** - MongoDB - Redis
Body Celkem	**4**	**5**	**4**

Ze tabulky 1 výše vyplývá, že ze srovnání nejlépe vyšel framework Botpress, ten bude tedy v práci dále použit.

4.3 Kontejner

V souvislosti s požadavkem na snadné škálování chatbota zdroje uvádějí výhody provozu chatbota v kontejnerech [1, 2]. Tento přístup bude použit i v této práci. Vzhledem k tomu,

že zdaleka nejpoužívanějším softwarem pro práci s kontejnery je software Docker [27, s. 7], bude tento použit pro práci s kontejnery i v této práci.

Kniha *Learning Docker* [28, s. 2] uvádí, že kontejner je softwarový balík, který obsahuje vše potřebné pro běh určitého počítačového programu. Kontejnery je možné chápat jako odlehčenou verzi virtuálních serverů. Je možné je využívat obdobně, ale jsou méně náročné na systémové prostředky než plnohodnotné virtuální servery. Neběží v nich totiž zpravidla plnohodnotný operační systém zcela izolovaný od toho hostitelského, ale jsou využívány části hostitelského systému (například jádro). Také na rozdíl od virtuálních serverů nedochází k virtualizaci hardwaru, která má negativní dopad na výkon [28, str. 8].

Kontejnery jsou dále velmi vhodné pro provoz škálovatelných aplikací, protože jejich vytvoření a spuštění je několikanásobně rychlejší než vytvoření a spuštění virtuálních serverů. Spuštění většiny kontejnerů trvá méně než sekundu [29, str. 8].

V této práci budou Docker kontejnery využity právě pro provoz chatbota a jeho škálování.

4.4 Kubernetes

Po učinění rozhodnutí provozovat aplikaci v kontejnerech je nutné zvolit způsob, jakým budou kontejnery spravovány. Zejména je potřeba mít možnost z jednoho místa ovládat počet spuštěných kontejnerů napříč jednotlivými servery a také kontrolovat, že například v případě selhání jednoho kontejneru bude spuštěn kontejner nový.

V této práci bude použit nejpoužívanějším software pro zajištění tohoto Kubernetes [30, s. 256]. Ten umožňuje spustit několik instancí aplikace běžící v kontejneru. Zároveň umožňuje toto dělat napříč několika fyzickými servery a docílit tak vysoké dostupnosti celé platformy [31, s. 173].

Zároveň je možné se k dané aplikaci připojit přes jedinou adresu a Kubernetes podle definovaných pravidel bude rozdělat provoz mezi jednotlivé instance běžící v kontejnerech. Rovněž je možné regulovat počet běžících instancí podle současného zatížení platformy [32].

Toto je z hlediska efektivity provozu velmi výhodné, protože v každý okamžik je spuštěno jen takové množství instancí, které je nutné k zajištění stabilního chodu platformy, a není tak plýtváno prostředky na instance, které by zůstaly nevyužity. Zároveň ale máme jistotu, že v případě potřeby bude odpovídající množství instancí automaticky přidáno [33, s. 171].

Kubernetes obsahuje mechanismy pro kontrolování funkčnosti jednotlivých instancí, na základě kterých umí odebírat z clusteru instance, které z nějakého důvodu přestaly fungovat. Také se může pokusit o jejich restartování. Celý mechanismus kontroly umožňuje mít pro každou provozovanou aplikaci definované vlastní kroky k ověření její funkčnosti a mít tak zajištěno pro každou aplikaci monitorování na míru [33, s. 101].

Všechny tyto vlastnosti dělají z Kubernetes systém vhodný pro nasazení komplexních aplikací využívaných velkým množstvím uživatelů. Takto nasazené systémy lze totiž

prakticky neomezeně horizontálně škálovat a mají v sobě samoopravné mechanismy, které zajistí, že systém zůstane funkční, i když některé jeho komponenty přestanou fungovat [33, s. 232].

Velmi oblíbená pro provoz v Kubernetes je architektura mikroslužeb, kdy je výsledná aplikace složena z jednoúčelových funkcí, které jsou poskytovány v kontejnerech. Velkou výhodou je, že komponenty takto sestavených aplikací jsou na sobě nezávislé a přidávání nebo odebírání instancí jednotlivých funkcí je tak velmi snadné [34, s. 2-4].

Celý systém je navíc velmi flexibilní, protože v každém kontejneru lze podle potřeby provozovat jiný operační systém a lze tak snadno nasadit velké množství aplikací, které potřebují k fungování nějaký druh specifického softwaru [35].

Pro další části práce je nutné se seznámit s pojmy „Kubernetes Master“ a „Kubernetes Node“ [36].

- **Kubernetes Master** – Server, který slouží ke správě všech kontejnerů a dalších objektů, které v rámci Kubernetes existují.
- **Kubernetes Node** – Server, na kterém jsou spuštěny kontejnery s aplikacemi. Ty jsou ovládány z Kubernetes Masteru.

4.5 Databáze

Pro provoz chatbota, který bude splňovat parametry popsané ve třetí kapitole, je nutné zvolit odpovídající databázový systém. Pro popisovaný prototyp bude v rámci práce vybrán takový, který splňuje požadavky kladené na samotného chatbota. Musí umožňovat vysoce dostupné nasazení a musí být škálovatelný.

K vysoké dostupnosti a škálovatelnosti databázových systémů lze v principu přistupovat dvěma způsoby. V prvním případě mohou být nástroje k jejich zajištění přímo součástí databázového systému. To znamená, že databázový systém stačí vhodně nakonfigurovat a požadované parametry jsou splněny [37, s. 252-254].

V druhém případě lze dosáhnout vysoké dostupnosti a škálovatelnosti dosáhnout za pomoci dalších nástrojů. Výhodou tohoto přístupu je zejména to, že zpravidla nabízí větší možnosti pro návrh výsledné architektury systému a není omezen na konfigurace, se kterými počítali autoři databázového systému. Je tedy možné si sestavit řešení na míru potřebám provozované aplikace [21, s. 103-112].

Naopak nevýhodou je, že tento způsob je náročnější na konfiguraci, nemusí být databázovým systémem oficiálně podporován a může např. dojít k nekompatibilitě při aktualizaci databázového systému. Proto je nutné takové konfigurace velmi důsledně testovat a při každé změně důkladně ověřit, že změny ve fungování databázového systému odpovídají očekáváním [38, s. 692-698].

Pro účely této práce bude považováno za výhodu, pokud bude zkoumaný databázový systém podporovat nastavení vysoké dostupnosti a škálovatelnosti bez použití dalších nástrojů,

protože práce není zaměřena na zkoumání možných konfigurací databázových systémů. Nicméně ale je jejím cílem popsat takové nasazení chatbota, které bude vysoce dostupné a škálovatelné, a výběr a konfigurace vhodného databázového systému je důležitý krok, který nesmí být vynechán.

5 Tvorba Prototypu

V této kapitole je popsán postup, jakým je vytvářen prototyp chatbota, který si práce klade za cíl vytvořit.

5.1 Popis prostředí

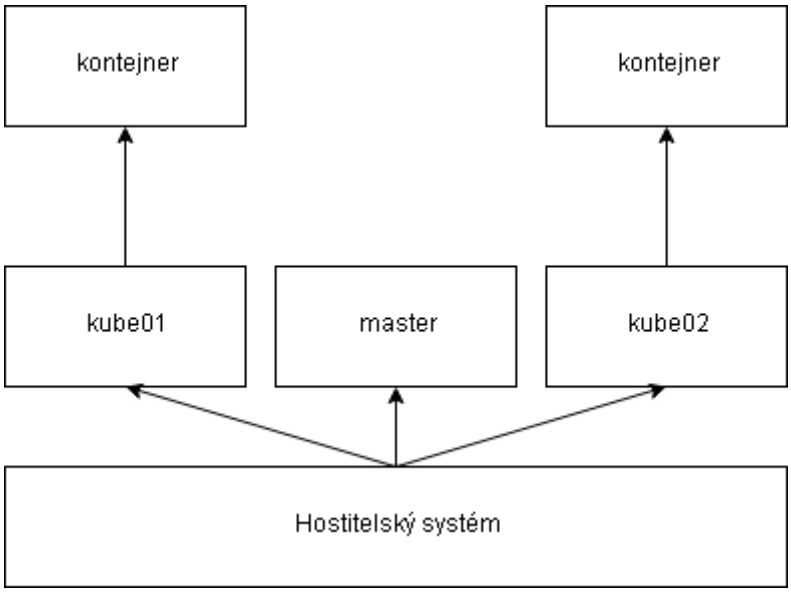
Prototyp bude vytvářen na virtuálních serverech „kube01“, „kube02“ a „master“. Server „master“ bude sloužit jako Kubernetes Master a zbývající dva servery budou mít roli Kubernetes Node. Všechny tři zmíněné servery budou mít následující konfiguraci:

- 2 CPU
- 3 GB RAM
- 10 GB SSD
- Centos 7

Tyto virtuální servery budou provozovány na fyzickém počítači s následující konfigurací:

- Intel(R) Core(TM) i7-6600U CPU
- 19 GB RAM
- 512 GB SSD
- Arch Linux

Servery budou virtualizovány pomocí nástroje Vagrant [39]. Na nich budou spuštěny kontejnery a v nich bude spuštěn chatbot (viz Obrázek 12).



Obrázek 12 Architektura prototypu [zdroj: autor]

5.2 Vytvoření virtuálních serverů

Virtuální servery budou vytvořeny pomocí nástroje pro tvorbu a správu virtuálních prostředí Vagrant. V jeho prostředí jsou definovány tři výše zmíněné virtuální servery, jsou jim přiřazeny IP adresy a hardwarová konfigurace popsaná v předchozí kapitole. Dále síťové porty 3000, 3001 a 3002 na hostitelském počítači jsou přesměrovány na port 31227 na virtuálních serverech. Port 31227 na virtuálních serverech je ty dostupný z hostitelského systému přes následující adresy (viz Tabulka 2).

Tabulka 2 Mapování portů [zdroj: autor]

Hostitelský systém	**Virtuální server**
localhost:3000	master:31227
localhost:3001	kube01:31227
localhost:3002	kube02:31227

Framework Botpress ve výchozím nastavení poslouchá na portu 3000 a v některých případech generuje odkazy, které v sobě tento port obsahují. Komunikace s chatbotem bude probíhat z hostitelského systému přes virtuální server „master“. Z tohoto důvodu je právě port 3000 na hostitelském systému přesměrován na server „master“. V případě použití jiného portu by bylo nutné použít externí nástroj na úpravu odkazů vygenerovaných chatbotem. Příkladem takového nástroje by mohl být webový server Apache fungující jako reverzní proxy s nastavenými pravidly pro přepisování URL.

Virtuální servery mají ve výchozím stavu síťová rozhraní eth0 a eth1. První rozhraní je použito pro přístup do internetu přes hostitelský systém a druhé je pak připojeno do sítě, po které všechny tři virtuální servery komunikují mezi sebou (viz Výpis 1).

Výpis 1 IP adresy serveru kube01 [zdroj: autor]

```
[root@kube02 ~]# ip a show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    group default qlen 1000link/ether 52:54:00:26:10:60 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global noprefixroute dynamic eth0
       valid_lft 85938sec preferred_lft 85938sec
    inet6 fe80::5054:ff:fe26:1060/64 scope link
       valid_lft forever preferred_lft forever
[root@kube02 ~]# ip a show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    group default qlen 1000
    link/ether 08:00:27:24:71:c6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.15.22/24 brd 10.0.15.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe24:71c6/64 scope link
       valid_lft forever preferred_lft forever
```

Na všech virtuálních serverech je vytvořen uživatelský účet „kube“, jehož pár SSH (Secure Shell) klíčů je přítomen rovněž na všech serverech a lze ho tedy využít ke vzdálenému spouštění příkazů a přenosu souborů bez nutnosti zadávání hesla. To bude později použito pro automatizaci nasazení Kubernetes clusteru.

Konfigurace operačního systému na virtuálních serverech je řešena skriptem (Příloha A), který Vagrant spustí při prvním spuštění každého serveru.

Konfigurační soubor Vagrantu je v práci přiložen jako Příloha B.

5.3 Databáze

Jako databázový systém pro Botpress byl vybrán databázový systém PostgreSQL. Ten je doporučovaný autory Botpressu a je ho možné provozovat ve vysoce dostupné konfiguraci a rovněž jej lze dobře škálovat [38, s. 7-17]. Tyto techniky nebudou v práci dále rozebírány, protože prototyp vytvořený v rámci této práce si klade za cíl zabývat se škálovatelností samotného chatbota. Škálování a vysoká dostupnost databázových systémů je samostatné téma, které s chatboty sice souvisí, ale není jejich přímou součástí.

Pro účely práce byl použit databázový systém PostgreSQL ve verzi 9.6.12 nainstalovaný na serveru dostupném z internetu.

5.4 Nasazení Kubernetes

Na použitém operačním systému Centos 7 je pro provozování Kubernetes nutné nainstalovat následující balíčky [40]:

- yum-utils
- device-mapper-persistent-data

- lvm2
- docker-ce
- kubelet
- kubeadm
- kubectl

Vzhledem k tomu, že všechny servery mají více než jednu IP adresu a síťové rozhraní, je nutné specifikovat, pod jakou IP adresou má každý server v rámci Kubernetes clusteru vystupovat. Pro tyto účely byla využita privátní síť 10.0.15.9/24, do které jsou servery připojeny přes síťové rozhraní eth1. Aby bylo tohoto docíleno, je potřeba zapsat do proměnné prostředí „KUBELET_EXTRA_ARGS“ IP adresu na rozhraní eth1 ve formátu „--node-ip=<IP adresa>“.

Pro správné fungování sítové komunikace na úrovni kontejnerů je potřeba na virtuálních serverech nastavit hodnotu /proc/sys/net/bridge/bridge-nf-call-iptables na 1 [41]. Tím je docíleno toho, že síťový provoz mezi virtuálními servery je zpracován programem iptables a jsou na něj aplikována pravidla vytvořená systémem Kubernetes. Ve výchozím nastavení operačního systému Centos 7 je tento druh síťového ze zpracování iptables vyňat.

Po úspěšném nainstalování všech komponent je nutné ze zamýšleného Kubernetes Masteru inicializovat cluster. Při inicializaci je opět nutné na příkazové řádce specifikovat, pod jakou IP adresou bude Kubernetes Master vystupovat a jaká síť má být použita pro provoz kontejnerů. Zde byla zvolena síť 10.244.0.0/16, která se pro tento účel často uvádí jako obvyklá [42]. Při inicializaci je na standartním výstupu zobrazen příkaz, který je nutné spustit na ostatních serverech, aby se mohly připojit do Kubernetes clusteru. Příkaz v sobě obsahuje bezpečnostní token, kterým se musí server při pokusu o připojení do Kubernetes clusteru prokázat.

V rámci zautomatizování celého procesu, tj. docílení toho, aby byl token předán automaticky a připojení serveru do Kubernetes clusteru nevyžadovalo žádnou manuální akci, je příkaz pro připojení, který je zobrazen při inicializaci Kubernetes clusteru, včetně bezpečnostního tokenu přečten ze standartního výstupu a zapsán do souboru (viz Výpis 2).

Výpis 2 Inicializace Kubernetes clusteru [zdroj: autor]

```
[root@master ~]# kubeadm init --apiserver-advertise-address=${ip} \
--pod-network-cidr=10.244.0.0/16 | grep -A1 '^kubeadm join' > /tmp/join_command

[root@master ~]# cat /tmp/join_command
kubeadm join 10.0.15.10:6443 --token zaadbh.ga2p9mkpnqjy8ow2 \
    --discovery-token-ca-cert-hash \
sha256:5552d653e3623bbe842e256bbc31ffa7632d429f02d8faca6cf3e120de8eab2a
```

Z tohoto souboru bude příkaz později přečten z ostatních serverů, které se díky němu připojí do clusteru.

Po úspěšné inicializaci clusteru je nutné zajistit, aby mezi jednotlivými kontejnery fungovala síťová komunikace. Pro Kubernetes jsou k dispozici rozšíření, která toto realizují. První volba pro tuto práci bylo v současnosti nejpopulárnější rozšíření flannel [43, 44], to se

ovšem nepodařilo úspěšně nakonfigurovat. Pravděpodobně byla na vině kombinace se síťováním ve Vagrantu, na internetu lze dohledat mnoho případů, kdy uživatelé hlásí problémy.

Dále bylo vyzkoušeno síťové rozšíření Weave [45]. To se ukázalo jako velmi dobrá volba, protože cluster s ním funguje bezchybně, lze ho nainstalovat jedním příkazem a pro využití pro účely této práce nevyžaduje žádnou další konfiguraci. Nainstalovat jej lze následujícím příkazem.

```
[root@master ~]# kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-
version=$(kubectl version | base64 | tr -d '\n')"
```

Tímto krokem je dokončena konfigurace Kubernates Masteru a je možné přistoupit ke konfiguraci zbylých dvou serverů kube01 a kube02, které budou mít roli Kubernetes Node. Na ty je při jejich prvním spuštění zkopírován a následně spuštěn soubor obsahující příkaz pro připojení do clusteru, který byl dříve vygenerován při inicializaci clusteru na Kubernetes Masteru (viz Výpis 3).

Výpis 3 Připojení serveru kube01 do Kubernetes clusteru [zdroj: autor]

```
[root@kube01 ~]# sudo -u kube scp -o StrictHostKeyChecking=no -o \
UserKnownHostsFile=/dev/null master:/tmp/join_command /tmp

[root@kube01 ~]# sh /tmp/join_command
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system
get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-
config-1.14" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...
This node has joined the cluster:

* Certificate signing request was sent to apiserver and a response was
received.

* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Na serveru „master“ je nyní možné zkontrolovat, že servery „kube01“ i „kube02“ se skutečně úspěšně připojily do Kubernetes clusteru (viz Výpis 4).

Výpis 4 Seznam serverů v Kubernetes clusteru [zdroj: autor]

```
[root@master ~]# kubectl get nodes
NAME     STATUS   ROLES   AGE   VERSION
kube02   Ready    <none>  4m52s v1.14.1
kube01   Ready    <none>  9m20s v1.14.1
master   Ready    master  13m   v1.14.1
```

Tímto krokem je dokončen Kubernetes cluster a je možné na něm nasazovat aplikace.

5.5 Nasazení chatbota

Vzhledem k plánovanému provozování chatbota v kontejnerech byl využit oficiální Docker kontejner, který tvůrci frameworku Botpress nabízejí [46]. Očekávání bylo, že tyto kontejnery provozované v Kubernetes clusteru budou data ukládat výhradně do společného databázového systému a zároveň si tímto způsobem budou předávat data mezi sebou. Bylo ovšem zjištěno, že při prvním spuštění Botpressu je nutné mj. vytvořit administrátorský účet a zadat jméno chatbota, který má být na frameworku spuštěn. Tyto informace ovšem nejsou zapisovány do databáze, ale na lokální souborový systém. Až po tomto kroku dojde k inicializaci databáze v databázovém systému.

To znamená, že kdyby byl pro sestavovaný prototyp využit oficiální Docker kontejner od tvůrců Botpressu, musela by být každá nově vytvořená instance chatbota inicializována tvorbou administrátorského účtu a zadáním informací o chatbotovi a až poté by chatbot začal komunikovat s databázovým systémem. To je velký problém, protože prototyp má mýt mj. snadno a rychle škálovatelný a je tedy nutné, aby nově vytvořené instance chatbota (kontejnery) byly schopné fungovat bezprostředně po startu. V úvahu přichází dvě řešení této situace.

Prvním možným řešením je spouštění Botpressu z kontejnerů ze sdíleného síťového souborového systému, což by zajistilo, že inicializace Botpressu se provede právě jednou a data na souborovém systému vygenerovaná během inicializace budou dostupná na všech kontejnerech. Toto řešení je sice poměrně jednoduše realizovatelné, například pomocí NFS (Network File System) úložiště [47, s. 161], ale do celého prototypu přidává další komponentu a potenciální zdroj problémů.

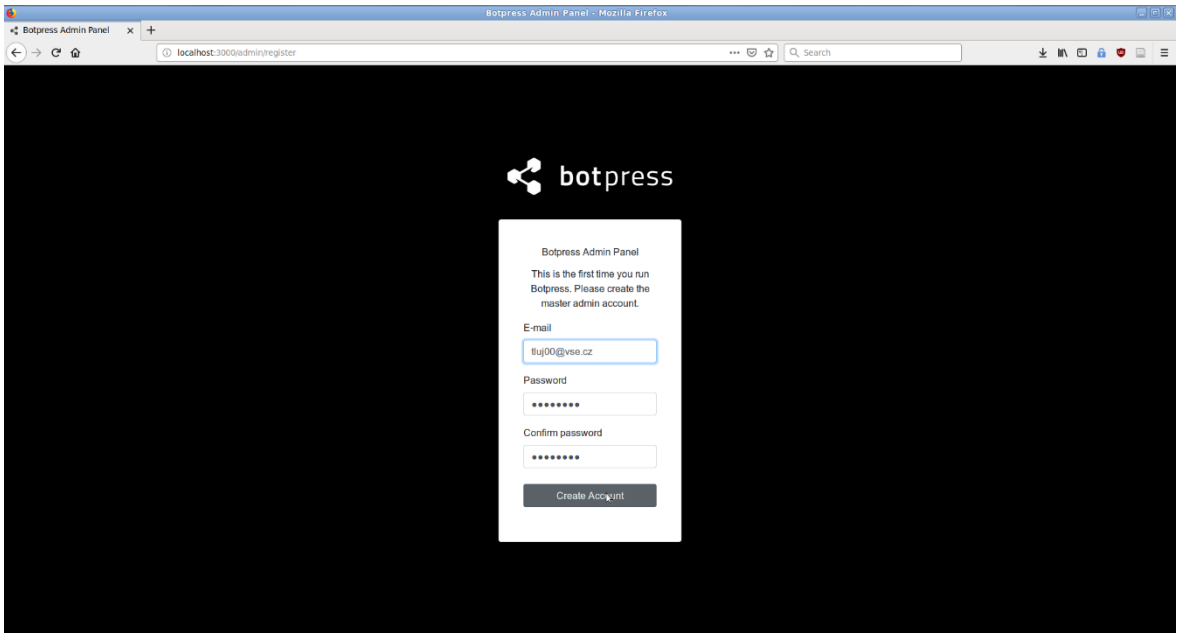
Druhou možností je sestavit takový kontejner, který bude již všechna potřebná data obsahovat a žádná inicializace nebude nutná. Kontejner tedy již v sobě musí obsahovat všechna data, která by za normálních okolností byla vytvořena při inicializaci Botpressu.

Pro další postup bylo zvoleno druhé řešení, protože na rozdíl od prvního nepřidává do prototypu další komponentu a nemění tak jeho zamýšlenou architekturu, ta po úpravě kontejneru zůstane zcela zachována.

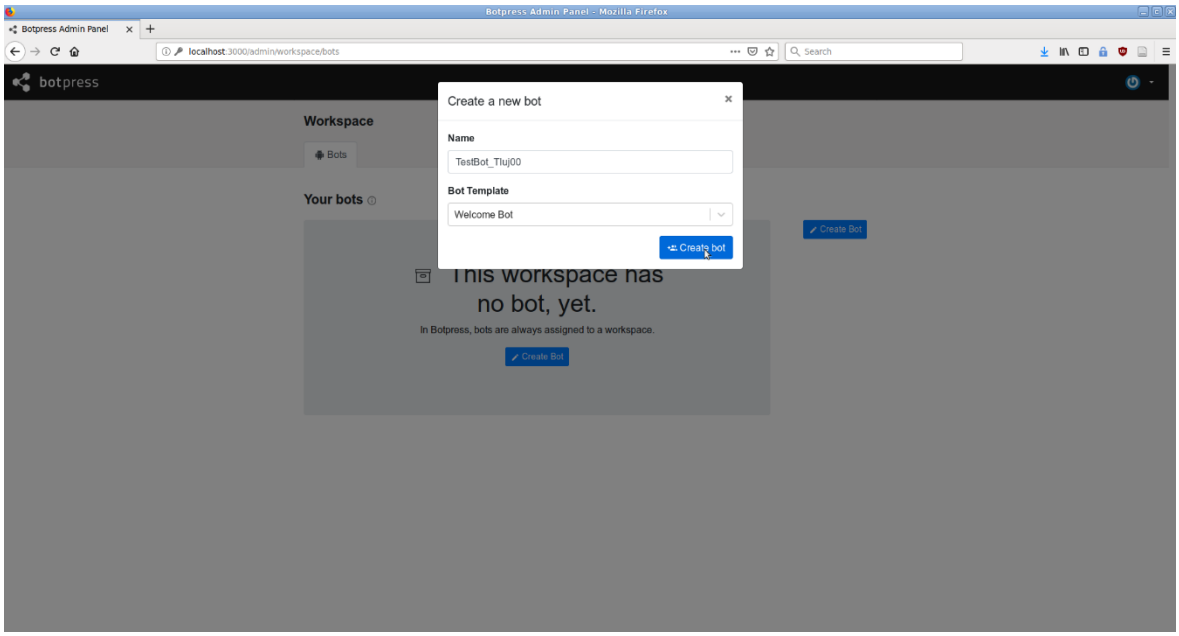
Postup úpravy kontejneru spočívá v jeho spuštění na lokální instalaci Dockeru (viz Výpis 5) a následném provedení inicializace Botpressu přes webový prohlížeč na adrese localhost:3000 (viz Obrázek 13 a 14).

Výpis 5 Spuštění Docker kontejneru [zdroj: autor]

```
[honza@honza-ntb kubernetes-botpress]$ docker run --detach \
> --name=botpress \
> --publish 3000:3000 \
> --volume botpress_data:/botpress/data \
> --env DATABASE=postgres \
> --env DATABASE_URL=postgres://botpress:botpass@207.154.207.148:5432/botpress\
> botpress/server:v11_7_4
4178e429aa208250caf00db9dd774df80a3acbdd09b97b42d8325f311acdb7f3
```



Obrázek 13 Botpress – vytvoření účtu [zdroj: autor]



Obrázek 14 Botpress – vytvoření chatbota [zdroj: autor]

Po vytvoření administrátorského účtu a chatbota jsou již na souborovém systému kontejneru uloženy všechny potřebné soubory a je možné zkontrolovat, že došlo k inicializaci databáze (viz Výpis 6).

Výpis 6 Obsah PostgreSQL databáze [zdroj: autor]

```
bash-4.2$ psql botpress
psql (9.2.24, server 9.6.12)
Type "help" for help.
botpress=# \dt
List of relations
Schema  |           Name             | Type  |  Owner
--------+----------------------------+-------+----------
public  | analytics_custom           | table | botpress
public  | analytics_interactions     | table | botpress
public  | analytics_runs             | table | botpress
public  | data_retention             | table | botpress
public  | dialog_sessions            | table | botpress
public  | knex_core_migrations       | table | botpress
public  | knex_core_migrations_lock  | table | botpress
public  | srv_channel_users          | table | botpress
public  | srv_ghost_files            | table | botpress
public  | srv_ghost_index            | table | botpress
public  | srv_kvs                    | table | botpress
public  | srv_logs                   | table | botpress
public  | srv_metadata               | table | botpress
public  | srv_notifications          | table | botpress
public  | web_conversations          | table | botpress
public  | web_messages               | table | botpress

(16 rows)
```

Nyní, když je chatbot inicializován, je potřeba z kontejneru zkopírovat nově vytvořené soubory.

```
[honza@honza-ntb kubernetes-botpress]$ docker cp e04349bdac5d:/botpress .
[honza@honza-ntb kubernetes-botpress]$ ls botpress
assets bindings bp data Dockerfile modules uploads
```

Je možné ověřit, že složka skutečně obsahuje potřebná data.

```
[honza@honza-ntb kubernetes-botpress]$ grep -r tluj00@vse.cz botpress
botpress/data/global/workspaces.json: "email": "tluj00@vse.cz",
botpress/data/global/botpress.config.json: "tluj00@vse.cz"
```

V tento okamžik je již k dispozici vše potřebné pro sestavení nového kontejneru. Software Docker pro tyto účely používá tzv. Dockerfile. Dockerfile je předpis, který popisuje kroky nutné k vytvoření konkrétního kontejneru [48]. Dockerfile použitý pro sestavení použitého Botpress kontejneru lze nalézt ve zkopírované složce (viz Výpis 7).

Výpis 7 Dockerfile [zdroj: autor]

```
[honza@honza-ntb kubernetes-botpress]$ cat botpress/Dockerfile
FROM ubuntu:18.04
ADD . /botpress
WORKDIR /botpress
RUN chmod +x bp
EXPOSE 3000
CMD ["./bp"]
```

První řádek Dockerfilu říká, že jako základ kontejneru má být použit již existující kontejner Ubuntu 18.04, který je volně dostupný na internetu. V druhém kroku je do existujícího kontejneru Ubuntu 18.04 přidána lokální složka „botpress“. Jako třetí krok je tato složka nastavena jako aktuální složka a jako čtvrtý krok je soubor „bp“ je označen jako spustitelný. V pátém kroku je specifikováno, že aplikace uvnitř kontejneru je dostupná na portu 3000. Na posledním řádku je nastaveno, že soubor „bp“ má být spuštěn při každém startu kontejneru.

Vzhledem k tomu, že jediná změna, kterou je nutné v kontejneru provést, je modifikace souborů ve složce „botpress“, aby odpovídaly stavu po inicializaci Botpressu, a vzhledem k tomu, že složka zkopírovaná z kontejneru již toto splňuje, je možné pro sestavení nového kontejneru použít nezměněný Dockerfile v této složce (viz Výpis 8).

Výpis 8 Sestavení upraveného Docker kontejneru [zdroj: autor]

```
[honza@honza-ntb botpress]$ docker build -t honzat/kubebot:v11_7_4 .
Sending build context to Docker daemon 386.9MB
Step 1/6 : FROM ubuntu:18.04
---> 94e814e2efa8
Step 2/6 : ADD . /botpress
---> c5126599c91a
Step 3/6 : WORKDIR /botpress
---> e2a646204e36
Step 4/6 : RUN chmod +x bp
---> 3688ced0d85d
Step 5/6 : EXPOSE 3000
---> 64df60166dca
Step 6/6 : CMD ["./bp"]
---> a1bd3c79e482
Successfully built a1bd3c79e482
Successfully tagged honzat/kubebot:v11_7_4
```

Kontejner je nyní hotový a je nutné ho připravit k distribuci na virtuální servery. Docker pro toto využívá repozitáře obsahující obrazy kontejnerů. Pro účely této práce byl vytvořen repozitář „honzat/kubebot“. Tento repozitář bude použit pro nahrání nově vytvořeného kontejneru. Nejprve je potřeba se přihlásit do předem registrovaného účtu na https://hub.docker.com/ (viz Výpis 9).

Výpis 9 Přihlášení na Docker Hub [zdroj: autor]

```
[honza@honza-ntb botpress]$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username:
Password:
WARNING! Your password will be stored unencrypted in
/home/honza/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

A následně je možné obraz kontejneru nahrát (viz Výpis 10).

Výpis 10 Nahrání upraveného kontejneru na Docker Hub [zdroj: autor]

```
[honza@honza-ntb botpress]$ docker push honzat/kubebot:v11_7_4
The push refers to repository [docker.io/honzat/kubebot]
4ced61e71126: Pushed
b57c79f4a9f3: Layer already exists
d60e01b37e74: Layer already exists
e45cfbc98a50: Layer already exists
762d8e1a6054: Layer already exists
v11_7_4_foo: digest:
sha256:4489f864cc899c7371512b3af5e73e3f523c61c81c6cb972a096e14c7560750f size:
1363
```

Když je obraz kontejneru nahrán do repozitáře, je možné přistoupit k jeho spuštění v Kubernetes. Nejprve je vytvořen Kubernetes deployment a na něm pak Kubernetes service. Na úrovni Kubernetes deploymentu je přes proměnné prostředí „DATABASE“ a „DATABASE_URL“ nakonfigurováno, aby Botpress nevyužíval jako úložiště databázový systém SQLite, což je jeho výchozí nastavení, ale aby namísto něj použil připravený PostgreSQL server (viz Výpis 11).

Výpis 11 Vytvoření Kubernetes deploymentu [zdroj: autor]

```
[root@master ~]# kubectl create deployment botpress \
--image=index.docker.io/honzat/kubebot:v11_7_4

[root@master ~]# kubectl patch deployment botpress -p \
'{"spec":{"template":{"spec":{"containers":[{"name":"kubebot","env":[{"name":"D
ATABASE","value":"postgres"},{"name":"DATABASE_URL","value":"postgres://botpres
s:botpass@207.154.207.148:5432/botpress"}]}]}}}}'
```

Deployment je nyní připravený a lze vytvořit Kubernetes service, se kterou lze komunikovat na portu 3000 kontejneru.

```
[root@master ~]# kubectl create service nodeport botpress --tcp=3000
```

Nyní je spuštěn kontejner a v něm Botpress nakonfigurovaný podle potřeb práce.

Na vytvořené Kubernetes service je ještě nutné provést úpravu konfigurace, aby bylo možné chatbota plnohodnotně používat.

```
[root@master ~]# kubectl patch svc botpress -p '{"spec": {"sessionAffinity": "ClientIP"}}'
[root@master ~]# kubectl patch svc botpress -p '{"spec": {"ports": [{"port":
3000,"nodePort": 31227,"name": "3000"}]}}'
```

Položka konfigurace „sessionAffinity“ je nastavena hodnotu „ClientIP“. To zabezpečí, že při komunikaci s chatbotem přes jeho vestavěné webové chatovací rozhraní bude komunikace probíhat se stejným kontejnerem, ze kterého bylo načteno webové rozhraní. Dále je nastaveno, že porty 3000 na kontejnerech budou na virtuálních serverech přístupné na portu 31227. Zvolený port musí být v rozsahu 30000-32767 [49].

V tuto chvíli už je Botpress spuštěn v kontejneru a je možné se k němu připojit. Je možné zobrazit spuštěné kontejnery (viz Výpis 12).

Výpis 12 Jeden spuštěný kontejner [zdroj: autor]

```
[root@master ~]# kubectl get pods
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-swf4b   1/1    Running   0          8m16s

[root@master ~]# for pod in $(kubectl get pods | awk '{print $1}' | grep -v \
'^NAME'); do kubectl describe pods $pod | grep '^Node:'; done
Node:   kube01/10.0.15.22
```

6 Ověření vlastností prototypu

V této kapitole bude ověřeno, že prototyp je škálovatelný a vysoce dostupný.

6.1 Škálovatelnost

Nyní je Botpress spuštěn v jednom kontejneru na serveru „kube01“. Chatbot má být vysoce dostupný a škálovatelný. Další instanci lze přidat jednoduše spuštěním následujícího příkazu.

```
[root@master ~]# kubectl scale --replicas=2 deployment/botpress
deployment.extensions/botpress scaled
```

Další kontejner je vytvořen (viz Výpis 13) a spuštěn na druhém serveru (viz Výpis 14).

Výpis 13 Dva spuštěné kontejnery [zdroj: autor]

```
[root@master ~]# kubectl get pods
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-nbblw   1/1    Running   0          2m50s
botpress-5f749bf574-swf4b   1/1    Running   0          24m
```

Výpis 14 Spuštěné kontejnery na serverech [zdroj: autor]

```
[root@master ~]# for pod in $(kubectl get pods | awk '{print $1}' | grep -v
'^NAME'); do kubectl describe pods $pod | grep '^Node:'; done
Node:   kube01/10.0.15.22
Node:   kube02/10.0.15.21
```

Je vidět, že nyní jsou spuštěny kontejnery dva. Nyní je možné ověřit, že provoz směřující na Botpress je dělen mezi oba dva. Pro tento účel lze použít nástroj tcpdump, který slouží k analyzování síťového provozu [50]. Ten v kontejnerech není nainstalovaný, protože u kontejnerů je většinou požadavek na ponechání jen takových součástí systémů, které jsou nezbytné pro bezproblémové fungování aplikace, která je ve formě kontejneru distribuována. Je tedy nutné se do kontejnerů přihlásit a nainstalovat jej.

```
[root@master ~]# kubectl exec -it botpress-5f749bf574-nbblw -- /bin/bash
root@botpress-5f749bf574-nbblw:/botpress# apt-get update && apt-get install \
tcpdump
```

Tato procedura je provedena na obou kontejnerech. Následně je možné tcpdump spustit, připojit se přes webový prohlížeč do Botpressu a sledovat síťový provoz na portu 3000. Na Výpisu 15 je spuštěn program tcpdump v kontejneru na serveru kube01 a na Výpisu 16 je spuštěn program tcpdump v kontejneru na serveru kube02.

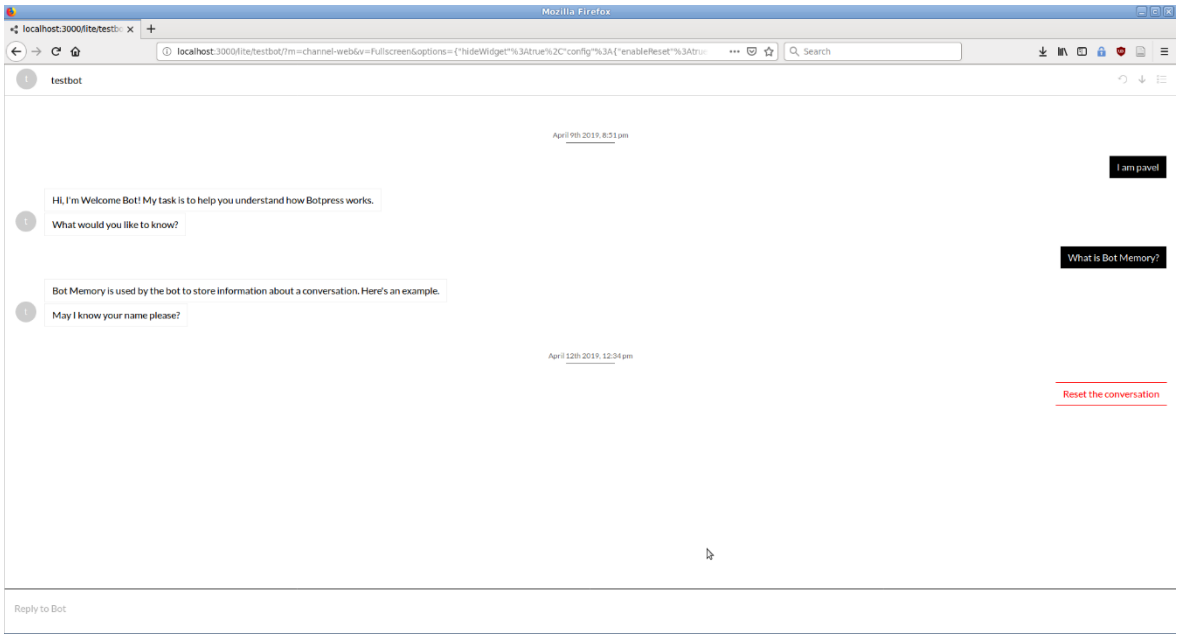
Výpis 15 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube01 [zdroj: autor]

```
[root@kube01 ~]# kubectl exec -it botpress-5f749bf574-nbblw -- /bin/bash
root@botpress-5f749bf574-nbblw:/botpress# tcpdump port 3000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Výpis 16 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube02 [zdroj: autor]

```
[root@kube02 ~]# kubectl exec -it botpress-5f749bf574-swf4b -- /bin/bash
root@botpress-5f749bf574-swf4b:/botpress# tcpdump port 3000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Nyní je možné se do Botpressu přihlásit (viz Obrázek 15) a sledovat síťový provoz na obou kontejnerech (viz Výpis 17 a 18).



Obrázek 15 Botpress – webový chat [zdroj: autor]

Výpis 17 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube01 – první test [zdroj: autor]

```
[root@kube01 ~]# kubectl exec -it botpress-5f749bf574-nbblw -- /bin/bash
root@botpress-5f749bf574-nbblw:/botpress# tcpdump port 3000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

06:28:12.791541 IP 10.32.0.1.38392 > botpress-5f749bf574-nbblw.3000: Flags [S],
seq 636544001, win 65535, options [mss 1460], length 0

06:28:12.791584 IP botpress-5f749bf574-nbblw.3000 > 10.32.0.1.38392: Flags
[S.], seq 842322183, ack 636544002, win 26720, options [mss 1336], length 0

06:28:12.792387 IP 10.32.0.1.38392 > botpress-5f749bf574-nbblw.3000: Flags [.],
ack 1, win 65535, length 0

06:28:12.793202 IP 10.32.0.1.38392 > botpress-5f749bf574-nbblw.3000: Flags
[P.], seq 1:359, ack 1, win 65535, length 358

06:28:12.793216 IP botpress-5f749bf574-nbblw.3000 > 10.32.0.1.38392: Flags [.],
ack 359, win 27336, length 0

06:28:12.800289 IP botpress-5f749bf574-nbblw.3000 > 10.32.0.1.38392: Flags
[P.], seq 1:309, ack 359, win 27336, length 308
```

Výpis 18 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube02 – první test [zdroj: autor]

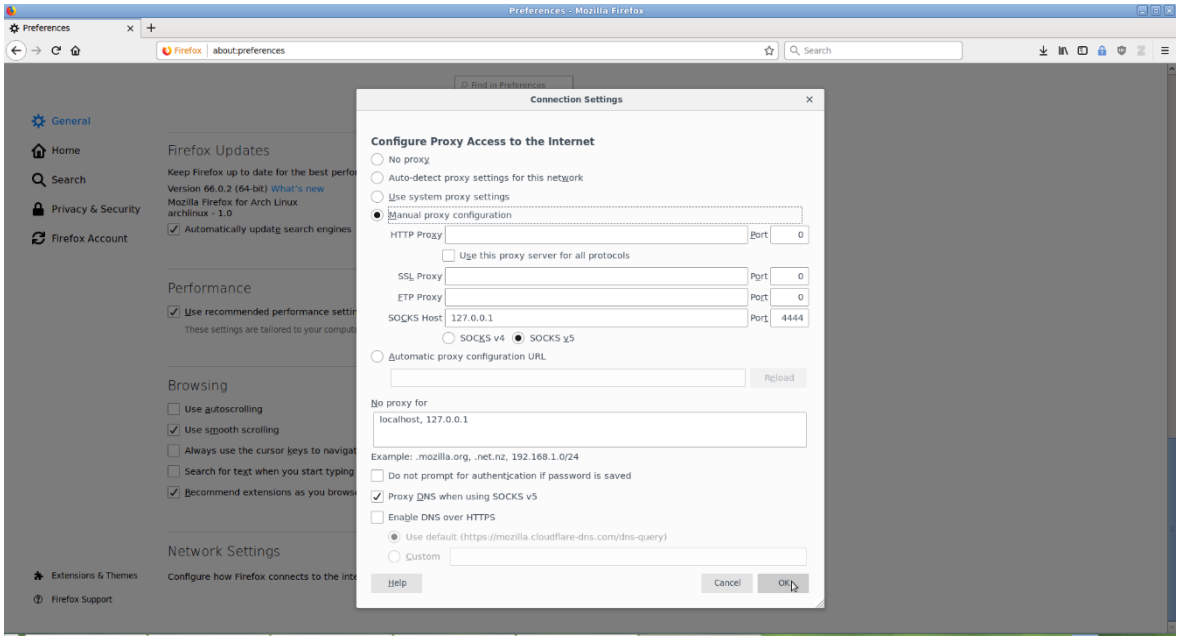
```
[root@kube02 ~]# kubectl exec -it botpress-5f749bf574-swf4b -- /bin/bash
root@botpress-5f749bf574-swf4b:/botpress# tcpdump port 3000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Provoz z fyzického počítače je tedy směrován na jeden kontejner, ten na serveru kube01. To je důsledek nastavení konfigurační položky „sessionAffinity“ na hodnotu „ClientIP". Nejsnazší způsob, jak ověřit funkčnost Botpressu v druhém kontejneru, je připojit se z jiné IP adresy. Dosud byla připojení do webového rozhraní Botpressu prováděna z hostitelského systému přes lokální port 3000 přesměrovaný na port 31227 na serveru „master“. Právě server „master“ lze využít jako druhý server pro přístup k Botpressu přes webový prohlížeč.

Na serveru „master“ ale není nainstalováno žádné grafické rozhraní a provozování běžných prohlížečů není možné. Lze ale přes SSH vytvořit tzv. SOCKS proxy, přes kterou může webový prohlížeč na hostitelském systému přistoupit do Botpressu, tak aby jako zdrojová adresa spojení pro Botpress vystupovala IP adresa serveru „master“.

```
[honza@honza-ntb kubernetes-botpress]$ ssh -p2222 -D 4444 vagrant@localhost
```

Na SOCKS proxy na hostitelském systému se lze připojit na portu 4444. Konfigurace webového prohlížeče Firefox je na Obrázku 16.



Obrázek 16 Firefox – nastavení SOCKS proxy [zdroj: autor]

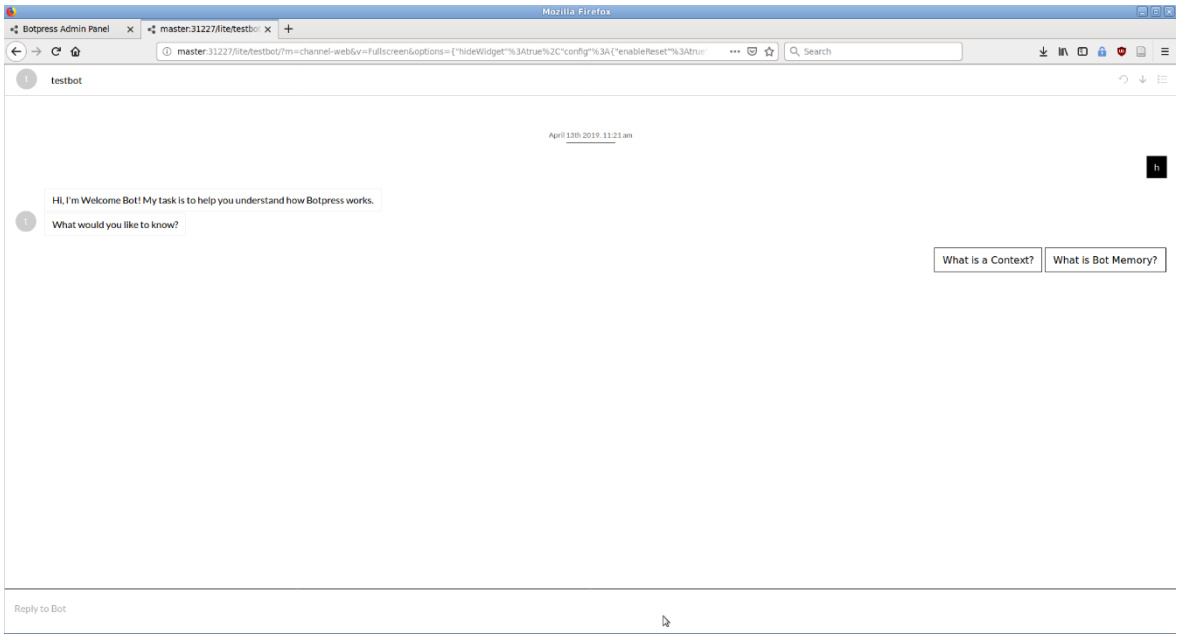
Je nutné dát pozor na to, aby i DNS (Domain Name System) dotazy byly posílány přes proxy server, v případě pokusu o lokální DNS překlad jména „master“ by došlo k chybě (viz Výpis 19).

Výpis 19 Neúspěšný DNS dotaz na doménové jméno „master“ [zdroj: autor]

```
[honza@honza-ntb kubernetes-botpress]$ nslookup master
Server: 10.0.0.138
Address: 10.0.0.138#53

** server can't find master: NXDOMAIN
```

Vše je nyní připravené a je možné se do Botpressu připojit přes připravenou SOCKS proxy (viz Obrázek 17).



Obrázek 17 Botpress – webový chat přes SOCKS proxy [zdroj: autor]

Ve spuštěném programu tcpdump je vidět, že v kontejneru na serveru kube01 není síťový provoz žádný a všechen je směrován do kontejneru na serveru kube02 (viz Výpis 20 a 21).

Výpis 20 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube01 – druhý test [zdroj: autor]

```
[root@kube01 ~]# kubectl exec -it botpress-5f749bf574-nbblw -- /bin/bash
root@botpress-5f749bf574-nbblw:/botpress# tcpdump port 3000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Výpis 21 Sledování síťového provozu na portu 3000 v kontejneru na serveru kube02 – druhý test [zdroj: autor]

```
[root@kube02 ~]# kubectl exec -it botpress-5f749bf574-swf4b -- /bin/bash
root@botpress-5f749bf574-swf4b:/botpress# tcpdump port 3000
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

09:17:44.939811 IP 10.32.0.1.47850 > botpress-5f749bf574-zwb7j.3000: Flags [S],
seq 3723024796, win 43690, options [mss 65495,sackOK,TS val 5956906 ecr
0,nop,wscale 7], length 0

09:17:44.939861 IP botpress-5f749bf574-zwb7j.3000 > 10.32.0.1.47850: Flags
[S.], seq 4038129479, ack 3723024797, win 26480, options [mss 1336,sackOK,TS
val 5798967 ecr 5956906,nop,wscale 7], length 0

09:17:44.943118 IP 10.32.0.1.47850 > botpress-5f749bf574-zwb7j.3000: Flags [.],
ack 1, win 342, options [nop,nop,TS val 5956910 ecr 5798967], length 0

09:17:44.944372 IP 10.32.0.1.47850 > botpress-5f749bf574-zwb7j.3000: Flags
[P.], seq 1:313, ack 1, win 342, options [nop,nop,TS val 5956912 ecr 5798967],
length 312

09:17:44.944404 IP botpress-5f749bf574-zwb7j.3000 > 10.32.0.1.47850: Flags [.],
ack 313, win 216, options [nop,nop,TS val 5798971 ecr 5956912], length 0

09:17:44.949774 IP botpress-5f749bf574-zwb7j.3000 > 10.32.0.1.47850: Flags
[P.], seq 1:309, ack 313, win 216, options [nop,nop,TS val 5798977 ecr
5956912], length 308
```

Je zřejmé, že prototyp lze úspěšně škálovat přidáváním kontejnerů, a bylo ověřeno, že tyto přidané kontejnery jsou při provozu Botpressu skutečně využity. Požadavek na škálovatelnost prototypu lze tedy považovat za splněný.

6.2 Vysoká dostupnost

Zde bude zkoumáno, jak bude Kubernetes cluster reagovat, když jeden z kontejnerů přestane fungovat. Očekávané chování je takové, že systém Kubernetes spustí kontejner nový a zajistí tak, že prototyp bude mít takovou architekturu, jaká byla zamýšlena.

Pro ověření tohoto chování je možné nasimulovat buď nečekané ukončení běhu kontejneru, aplikace uvnitř kontejneru nebo neplánované vypnutí jednoho ze serverů, na kterém je spuštěn kontejner. Rozebrány budou všechny tři zmíněné možnosti.

Kubernetes cluster by také bylo možné rozšířit o další Kubernetes Master [51]. Tím by bylo docíleno i jeho vysoké dostupnosti. Tato práce se zabývá pouze vysokou dostupností aplikace chatbota na úrovni kontejnerů.

6.2.1 Ukončení aplikace v kontejneru a ukončení kontejneru

Simulování nečekaného ukončení aplikace v kontejneru a kontejneru se v tomto případě překrývá, protože Botpress je v kontejneru spuštěn jako proces s PID (Process Identification Number) 1. To znamená, že jeho ukončení je ekvivalentem ukončení běhu kontejneru [52, s. 264].

```
root@botpress-5f749bf574-nbblw:/botpress# ps aux | grep bp
root 1 0.8 6.3 1302700 118624 ? Ssl 06:37 0:08 ./bp
root 31 0.0 0.0 11452 724 pts/0 S+ 06:54 0:00 grep --color=auto bp
```

To lze udělat ze serveru, na kterém běží kontejner, který má být ukončen. Procesy běžící v kontejneru jsou viditelné i na serveru, na kterém je spuštěn kontejner. Liší se PID, pod kterým proces vystupuje [52, s. 264].

Ověření chování bude probíhat na serveru kube01. Prvním krokem je identifikace procesu Botpressu.

```
[root@kube01 ~]# ps aux | grep bp
root 7432 3.6 6.0 1302188 114592 ? Ssl 06:37 0:05 ./bp
root 7617 0.0 0.0 12520 984 pts/0 S+ 06:39 0:00 grep --color=auto bp
```

PID procesu Botpressu je tedy 7432. Proces bude ukončen příkazem „kill -9“, ten je možné použít pro simulování neočekávaného ukončení programu [53, s. 143].

Kontejner byl ukončen v 07:13:52.

```
[root@kube01 ~]# date; kill -9 7432
Thu Apr 18 07:13:52 UTC 2019
```

Systém Kubernetes toto zaregistroval v 07:13:54. Zareagoval vytvořením nového kontejneru a v 07:14:07 byl obnoven žádoucí stav clusteru, tj. dva spuštěné kontejnery s aplikací. Celý proces zotavení tedy trval 15 sekund (viz Výpis 22).

Výpis 22 Test obnovy po ukončení kontejneru [zdroj: autor]

```
[root@master ~]# while true; do date; kubectl get pods; sleep 1; done
Thu Apr 18 07:13:53 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-nbblw   1/1    Running   1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:13:54 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:13:55 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:13:56 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:13:57 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:13:58 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:13:59 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:14:00 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:14:02 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:14:03 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:14:04 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:14:05 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    Error     1          36m
```

```
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
Thu Apr 18 07:14:06 UTC 2019
NAME                        READY  STATUS             RESTARTS   AGE
botpress-5f749bf574-46sgj   0/1    CrashLoopBackOff   1          36m
botpress-5f749bf574-zwb7j   1/1    Running            13         4d23h
Thu Apr 18 07:14:07 UTC 2019
NAME                        READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-46sgj   1/1    Running   2          36m
botpress-5f749bf574-zwb7j   1/1    Running   13         4d23h
```

Pro vyšší důvěryhodnost naměřených hodnot byl celý proces ještě třikrát zopakován, aby bylo možné naměřené hodnoty porovnat (viz Tabulka 3).

Tabulka 3 Test obnovy po ukončení kontejneru [zdroj: autor]

	Čas ukončení kontejneru	Čas zaznamenání události systémem Kubernetes	Čas obnovení žádoucího stavu	Celková doba obnovy
Test 1	07:13:52	07:13:54	07:14:07	15 sekund
Test 2	07:21:14	07:21:17	07:21:30	16 sekund
Test 3	07:23:35	07:23:38	07:23:51	16 sekund
Test 4	07:25:09	07:25:11	07:25:24	15 sekund

Je tedy vidět, že chování systému Kubernetes je napříč jednotlivými testy konzistentní a v tomto ohledu prototyp splňuje chování, které je od něj očekávané.

6.2.2 Vypnutí serveru

Zde bude provedena simulace nečekaného výpadku jednoho ze serverů, na kterém je spuštěn kontejner s Botpressem. Bude simulována situace, kdy je zapnutý server odpojen od zdroje napájení. Toho lze v programu Vagrant docílit zadáním příkazu „vagrant halt --force“ [54]. Očekávané chování programu Kubernetes je, že po vypnutí jednoho ze serverů se spuštěným kontejnerem spustí další kontejner na druhém ze serverů, aby byly zachovány takové parametry prototypu, jaké byly zamýšleny.

Server kube01 byl vypnut v 07:11:35.

```
 [honza@honza-ntb kubernetes-botpress]$ date; vagrant halt --force kube01
Sat Apr 20 07:11:35 CEST 2019
==> kube01: Forcing shutdown of VM...
```

Z Výpisu 23 je vidět, že systém Kubernetes toto zaznamenal v 07:12:14. V 07:17:20 pak přistoupil k vytvoření nového kontejneru a v 07:17:21 byl nový kontejner již spuštěn a připraven k použití. Celý proces navrácení k původnímu počtu spuštěných kontejnerů tedy trval 5 minut a 46 sekund. V 07:17:29 byl server kube01 opět spuštěn.

Na první pohled je pravděpodobně nápadné, že doba, která uplynula mezi okamžikem, kdy systém Kubernetes zaznamenal odpojení serveru kube01, a okamžikem, kdy vytvořil a spustil nový kontejner, je relativně dlouhá (5 minut a 6 sekund). To je zapříčiněno konfigurační položkou „pod-eviction-timeout“, jejíž defaultní hodnota je nastavena na pět minut.

Tato konfigurační položka udává, jak dlouho má systém Kubernetes po odpojení serveru počkat, než kontejnery, které byly na odpojeném serveru spuštěny, přesune na jiný server. Tuto hodnotu lze upravovat tak, aby odpovídala potřebám jednotlivých projektů. Pro účely této práce je považováno za dostatečné, že byla ověřena funkčnost tohoto mechanismu pro vytvořený prototyp a nový kontejner byl skutečně spuštěn.

Výpis 23 Test obnovy po vypnutí serveru [zdroj: autor]

```
[root@master ~]# while true; do date; kubectl get nodes; kubectl get pods;
sleep 1; done
Sat Apr 20 07:11:34 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   Ready    <none>  6d21h v1.14.1
kube02   Ready    <none>  6d21h v1.14.1
master   Ready    master  6d21h v1.14.1
NAME                         READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Running   0          83s
botpress-5f749bf574-zwb7j   1/1    Running   14         6d21h
Sat Apr 20 07:11:35 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   Ready    <none>  6d21h v1.14.1
kube02   Ready    <none>  6d21h v1.14.1
master   Ready    master  6d21h v1.14.1
NAME                         READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Running   0          84s
botpress-5f749bf574-zwb7j   1/1    Running   14         6d21h

<výstup zkrácen>

Sat Apr 20 07:12:14 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   NotReady <none>  6d21h v1.14.1
kube02   Ready    <none>  6d21h v1.14.1
master   Ready    master  6d21h v1.14.1
NAME                         READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Running   0          2m3s
botpress-5f749bf574-zwb7j   1/1    Running   14         6d21h
Sat Apr 20 07:12:15 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   NotReady <none>  6d21h v1.14.1
kube02   Ready    <none>  6d21h v1.14.1
master Ready master 6d21h v1.14.1
NAME                         READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Running   0          2m4s
botpress-5f749bf574-zwb7j   1/1    Running   14         6d21h

<výstup zkrácen>

Sat Apr 20 07:17:20 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   NotReady <none>  6d21h_v1.14.1
kube02   Ready    <none>  6d21h_v1.14.1
master   Ready    master  6d21h_v1.14.1
NAME                         READY  STATUS              RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Terminating         0          7m9s
botpress-5f749bf574-rwbwc   0/1    ContainerCreating   0          1s
```

```
botpress-5f749bf574-zwb7j   1/1    Running             14         6d21h
Sat Apr 20 07:17:21 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   NotReady <none>  6d21h v1.14.1
kube02   Ready    <none>  6d21h v1.14.1
master   Ready    master  6d21h v1.14.1
NAME                         READY  STATUS        RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Terminating   0          7m10s
botpress-5f749bf574-rwbwc   1/1    Running       0          2s
botpress-5f749bf574-zwb7j   1/1    Running       14         6d21h

<výstup zkrácen>

Sat Apr 20 07:17:49 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   Ready    <none> 6d21h  v1.14.1
kube02   Ready    <none> 6d21h  v1.14.1
master   Ready    master 6d21h  v1.14.1
NAME                         READY  STATUS        RESTARTS   AGE
botpress-5f749bf574-659fz   1/1    Terminating   0          7m38s
botpress-5f749bf574-rwbwc   1/1    Running       0          30s
botpress-5f749bf574-zwb7j   1/1    Running       14         6d21h
Sat Apr 20 07:17:50 UTC 2019
NAME     STATUS   ROLES   AGE   VERSION
kube01   Ready    <none>  6d21h v1.14.1
kube02   Ready    <none>  6d21h v1.14.1
master   Ready    master  6d21h v1.14.1
NAME                         READY  STATUS    RESTARTS   AGE
botpress-5f749bf574-rwbwc   1/1    Running   0          31s
botpress-5f749bf574-zwb7j   1/1    Running   14         6d21h
```

I zde byl test pro vyšší důvěryhodnost naměřených hodnot ještě třikrát zopakován (viz Tabulka 4).

Tabulka 4 Test obnovy po vypnutí serveru [zdroj: autor]

	Čas vypnutí serveru	Čas zaznamenání události systémem Kubernetes	Čas obnovení žádoucího stavu	Celková doba obnovy
Test 1	07:11:35	07:12:14	07:17:21	5 minut 46 sekund
Test 2	07:25:02	07:25:43	07:30:49	5 minut 47 sekund
Test 3	07:32:11	07:32:51	07:37:57	5 minut 46 sekund
Test 4	07:53:09	07:53:50	07:58:58	5 minut 49 sekund

Je vidět, že i v tomto případě je chování prototypu konzistentní a výsledky jednotlivých testů jsou velmi podobné. Lze prohlásit, že i zde jsou požadavky kladené na prototyp splněny, protože v případě ztráty jednoho ze serverů, na kterém je spuštěn kontejner, je tento kontejnery systémem Kubernetes realokován na jiný server.

7 Zhodnocení výsledků

V předchozí kapitole bylo předvedeno, že prototyp sestavený v rámci této diplomové práce splňuje požadavky na vysokou dostupnost a škálovatelnost, které na něj byly kladeny. Zároveň se podařilo celou infrastrukturu nutnou pro jeho provoz provozovat na jediném fyzickém počítači.

Pro jeho otestování stačí pouze naklonovat připravený repozitář [55] z internetu, spustit připravený skript a prototyp je připraven k použití. Lze jej tedy velmi snadno kdekoliv zprovoznit a dále používat pro práci na prototypu anebo použít Kubernetes cluster, který byl v rámci práce nakonfigurován, k vlastním účelům. To může ušetřit nemalé množství času, které by bylo na vytvoření vlastního lokálního Kubernetes clusteru zapotřebí. Vytvoření Kubernetes clusteru je plně automatizované a shellový skript, který toto zajišťuje, lze s pouze minimálními úpravami využít na serverech s nainstalovaným operačním systémem Centos 7 i mimo virtuální servery vytvořené v programu Vagrant v rámci této diplomové práce.

Prototyp ukazuje, že použití kontejnerů pro aplikace chatbotů je velmi vhodné a v kombinaci s nástrojem Kubernetes lze sestavit funkční, flexibilní a relativně snadno nasaditelná řešení.

Alternativa v podobě provozování chatbota na fyzických nebo virtuálních serverech by sice měla zdánlivou výhodu v menší komplexnosti architektury, ze které by byla odstraněna celá jedna vrstva v podobě kontejnerů, ale toto je vyváženo jednoduchostí, se kterou lze vytvořený prototyp škálovat. Použití kontejnerů pro podobné účely lze tedy jednoznačně doporučit.

Další výhodou jsou integrované mechanismy systému Kubernetes, které zaručují vysokou dostupnost. Té by v případě využití serverů muselo být dosaženo jinými nástroji a k přidání další komponenty do architektury prototypu by tedy došlo také.

Dojem z práce s programy Docker a Kubernetes je, že přes jejich zdánlivou prvotní přílišnou komplexnost a náročnost na pochopení se jedná o nástroje, jejichž použití je po pochopení jejich základních principů intuitivní a přímočaré. Zároveň okolo obou na internetu existují velmi početné komunity a při řešení konkrétních technických problémů je k dispozici velké množství materiálů.

Jediná složitější část je nasazení systému Kubernetes, to je dáno zejména jeho rychlým vývojem a také poměrně vysokou roztříštěností Linuxových distribucí. To má v praxi za důsledek, že velké množství dostupné dokumentace a manuálů není zcela aktuální. Ovšem ve většině případů se jedná pouze o drobnosti, které není těžké dohledat a upravit podle potřeb.

Podobný je i dojem z práce s nástrojem Vagrant. Jeho konfigurace je velmi přímočará. Zároveň ale mohou být vytvořené konfigurace velmi komplexní, Vagrant využívá pro

konfigurační soubory jazyk Ruby a lze v nich využívat jeho vestavěné konstrukce jako např. cykly. Konfigurovat lze nejenom virtuální počítače ale i sítě.

Prototyp využívá pouze open source nástroje a ukazuje, že je možné s jejich pomocí vytvořit a provozovat plnohodnotnou vysoce dostupnou a škálovatelnou aplikaci chatbota s širokými možnostmi konfigurace.

Závěr

Tato diplomová práce se zabývá provozními aspekty chatbotů. Nejdříve je provedena rešerše zdrojů zabývajících se tímto tématem. V druhé kapitole je úvod do problematiky chatbotů, je stručně popsán jejich historický vývoj a jsou zmíněny někteří významní chatboti, kteří byli v minulosti vytvořeni. Také jsou popsány základní způsoby, jakými mohou chatboti fungovat a zpracovávat vstup od uživatelů. Dále jsou popsány pojmy „vysoká dostupnost“ a „škálovatelnost“, jsou uvedeny jejich definice a specifika v oblasti chatbotů.

Další část diplomové práce je věnována vlastnímu prototypu chatbota. Ten je vytvořen výhradně pomocí open source softwarových nástrojů. Klade si za cíl být vysoce dostupný a snadno škálovatelný.

Při popisu jeho tvorby jsou nejprve vybrány a následně představeny vhodné frameworky, které by mohly být pro prototyp použity. Jsou stanovena kritéria pro jejich srovnání a následně je pro prototyp chatbota zvolen framework s nejlepším hodnocením. Dále jsou představeny a popsány další nástroje a technické principy, které jsou pro tvorbu prototypu použity.

Následuje popis vlastního procesu tvorby prototypu. Nejprve je popsáno virtualizované prostředí, na kterém prototyp vzniká. Po popisu prostředí je popsán proces nasazování systému Kubernetes, úprava kontejnerů a práce s nimi. Posledním popsaným krokem při tvorbě prototypu je nasazení nově vytvořeného kontejneru, ve kterém je spuštěna aplikace chatbota.

V šesté kapitole je ověřena funkčnost celého prototypu se zaměřením na jeho dostupnost a škálovatelnost. Je také popsáno sledování síťového provozu v kontejnerech pomocí nástroje tcpdump.

V poslední sedmé kapitole jsou shrnuty výsledky práce na prototypu a jsou prezentovány některé poznatky z jeho tvorby.

Naplnění cílů

Cíl sestavení vlastního prototypu chatbota za použití open source softwarových nástrojů byl splněn. Bylo také ověřeno, že prototyp je snadno škálovatelný a vysoce dostupný, s výjimkou serveru v roli Kubernetes Master.

Dílčí cíl porovnání tří nejoblíbenějších open source frameworků pro tvorbu chatbotů byl splněn ve čtvrté kapitole v rámci výběru vhodných komponent pro prototyp.

Přínosy práce

Přínosem práce je zejména zveřejněný repozitář s vytvořeným prototypem, který si může kdokoliv naklonovat z internetu a jedním příkazem prototyp spustit. Repozitář v sobě také obsahuje skript, který plně automatizuje vytvoření Kubernetes clusteru na serverech s operačním systémem Centos 7. Takto vytvořený cluster lze využít k libovolnému účelu.

Možnosti rozšíření práce

Na práci lze navázat doplněním prototypu o další Kubernetes Master a docílením tak i jeho vysoké dostupnosti. Další možností, jak navázat na tuto diplomovou práci, je nasazení prototypu do některé z dostupných cloudových platforem, jako je např. AWS.

Použitá literatura

1 YAN, Mengting, Paul CASTRO, Perry CHENG a Vatche ISHAKIAN. Building a Chatbot with Serverless Computing. In: *the 1st International Workshop*: *Proceedings of the 1st International Workshop on Mashups of Things and APIs - MOTA '16* [online]. Trento, Italy: ACM Press, 2016, s. 1–4 [vid. 2019-03-15]. ISBN 978-1-4503-4669-6. Dostupné z: doi:10.1145/3007203.3007217

2 ADEWALE, Oluwatosin, Alex BEATSON, Davit BUNIATYAN, Jason GE, Mikhail KHODAK, Holden LEE, Niranjani PRASAD, Nikunj SAUNSHI, Ari SEFF, Karan SINGH, Daniel SUO, Cyril ZHANG a Sanjeev ARORA. *Pixie: A Social Chatbot* [online]. B.m.: Department of Computer Science, Princeton University. 2017. Dostupné z: http://alexaprize.s3.amazonaws.com/2017/technical-article/pixie.pdf

3 D'SILVA, Godson Michael, Sanket THAKARE a Vinayak Ashok BHARADI. Real-time processing of IoT events using a Software as a Service (SaaS) architecture with graph database. In: *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*: *2016 International Conference on Computing Communication Control and automation (ICCUBEA)* [online]. Pune, India: IEEE, 2016, s. 1–6 [vid. 2019-03-15]. ISBN 978-1-5090-3291-4. Dostupné z: doi:10.1109/ICCUBEA.2016.7859984

4 ABASHEV, Almir, Ruslan GRIGORYEV, Karen GRIGORIAN a Vladislav BOYKO. Programming Tools for Messenger-Based Chatbot System Organization: Implication for Outpatient and Translational Medicines. *BioNanoScience* [online]. 2017, **7**(2), 403–407. ISSN 2191-1649. Dostupné z: doi:10.1007/s12668-016-0376-9

5 KHAN, Rashid a Anik DAS. *Build Better Chatbots* [online]. Berkeley, CA: Apress, 2018 [vid. 2019-03-15]. ISBN 978-1-4842-3110-4. Dostupné z: doi:10.1007/978-1-4842-3111-1

6 JANARTHANAM, Srini. *Hands-On Chatbots and Conversational UI Development: Build chatbots and voice user interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills*. 1 edition. B.m.: Packt Publishing, 2017.

7 MOOR, James H., ed. *The Turing Test* [online]. Dordrecht: Springer Netherlands, 2003 [vid. 2019-04-21]. Studies in Cognitive Systems. ISBN 978-1-4020-1205-1. Dostupné z: doi:10.1007/978-94-010-0105-2

8 JOSEPH, Lentin. *ROS Robotics Projects*. B.m.: Packt Publishing Ltd, 2017. ISBN 978-1-78355-472-0.

9 ROBINO, Giorgio. Bots builders: drag&drop, NLP or scripting? *Giorgio Robino* [online]. 31. prosinec 2016 [vid. 2019-04-24]. Dostupné z: https://medium.com/@solyarisoftware/bot-developement-gui-rad-nlp-or-scripting-f9bc7865b013

10 *Travel Insurance Chatbot – Zumata* [online]. [vid. 2019-04-24]. Dostupné z: https://zumata.com/ai-chatbots/chatbot-case-studies/travel-insurance-chatbot/

11 THANAKI, Jalaj. *Machine Learning Solutions: Expert techniques to tackle complex machine learning problems using Python*. B.m.: Packt Publishing Ltd, 2018. ISBN 978-1-78839-889-3.

12 KOMPELLA, Ravindra. Conversational AI chat-bot — Architecture overview. *Towards Data Science* [online]. 9. únor 2018 [vid. 2019-04-25]. Dostupné z: https://towardsdatascience.com/architecture-overview-of-a-conversational-ai-chat-bot-4ef3dfefd52e

13 BATISH, Rachel. *Voicebot and Chatbot Design: Flexible conversational interfaces with Amazon Alexa, Google Home, and Facebook Messenger*. B.m.: Packt Publishing Ltd, 2018. ISBN 978-1-78913-688-3.

14 THANAKI, Jalaj. *Python Natural Language Processing*. B.m.: Packt Publishing Ltd, 2017. ISBN 978-1-78728-552-1.

15 GN, Chethan Kumar. NLP vs NLU vs NLG (Know what you are trying to achieve) NLP engine (Part-1). *Towards Data Science* [online]. 25. září 2018 [vid. 2019-04-25]. Dostupné z: https://towardsdatascience.com/nlp-vs-nlu-vs-nlg-know-what-you-are-trying-to-achieve-nlp-engine-part-1-1487a2c8b696

16 LEHVÄ, Jyri, Niko MÄKITALO a Tommi MIKKONEN. Case Study: Building a Serverless Messenger Chatbot. In: Irene GARRIGÓS a Manuel WIMMER, ed. *Current Trends in Web Engineering* [online]. Cham: Springer International Publishing, 2018 [vid. 2019-03-17], s. 75–86. ISBN 978-3-319-74432-2. Dostupné z: doi:10.1007/978-3-319-74433-9_6

17 MARCUS, Evan a Hal STERN. *Blueprints for High Availability*. 2 edition. Indianapolis, Ind: Wiley, 2003. ISBN 978-0-471-43026-1.

18 WEINSTOCK, Charles B a John GOODENOUGH. *On System Scalability* [online]. B.m.: Carnegie Mellon University. 2006. Dostupné z: https://www.researchgate.net/publication/238687854_On_System_Scalability

19 BHOWMIK, Sandeep. *Cloud Computing*. B.m.: Cambridge University Press, 2017. ISBN 978-1-316-63810-1.

20 ZAMBRANO, Brian. *Serverless Design Patterns and Best Practices: Build, secure, and deploy enterprise ready serverless applications with AWS to improve developer productivity*. B.m.: Packt Publishing Ltd, 2018. ISBN 978-1-78862-438-1.

21 RESMAN, Mitja. *CentOS High Availability*. B.m.: Packt Publishing Ltd, 2015. ISBN 978-1-78528-532-5.

22 *Botkit: Building Blocks for Building Bots | Botkit Documentation* [online]. [vid. 2019-03-15]. Dostupné z: https://botkit.ai/docs/

23 *An open tool for designing, building and managing interactive dialog systems: howdyai/botkit-cms* [online]. JavaScript. B.m.: Howdy, 2019 [vid. 2019-04-25]. Dostupné z: https://github.com/howdyai/botkit-cms

24 *Botpress | Developer's Guide · Guides and references for all you need to know about Botpress* [online]. [vid. 2019-03-15]. Dostupné z: https://botpress.io/docs/index.html

25 ONIGBINDE, Ayomide. Building a TechCrunch bot clone using Botpress (Part 2). *Medium* [online]. 20. únor 2018 [vid. 2019-04-25]. Dostupné z: https://medium.com/@onigbindeayomidesnr/building-a-techcrunch-bot-clone-using-botpress-part-2-a56c1433f639

26 *Rasa Stack: Open source conversational AI* [online]. [vid. 2019-03-15]. Dostupné z: https://rasa.com/products/rasa-stack

27 ANTAO, Tiago. *Bioinformatics with Python Cookbook*. B.m.: Packt Publishing Ltd, 2015. ISBN 978-1-78355-865-0.

28 RAJ, Pethuru, Jeeva S. CHELLADHURAI a Vinod SINGH. *Learning Docker*. B.m.: Packt Publishing Ltd, 2015. ISBN 978-1-78439-193-5.

29 TURNBULL, James. *The Docker Book: Containerization Is the New Virtualization*. B.m.: James Turnbull, 2014. ISBN 978-0-9888202-0-3.

30 KLAFFENBACH, Florian, Oliver MICHALSKI, Markus KLEIN, Mohamed WALI, Namit TANASSERI a Rahul RAI. *Implementing Azure: Putting Modern DevOps to Use: Transform your software deployment process with Microsoft Azure*. B.m.: Packt Publishing Ltd, 2019. ISBN 978-1-78883-179-6.

31 SAITO, Hideto, Hui-Chuan Chloe LEE a Ke-Jou Carol HSU. *Kubernetes Cookbook*. B.m.: Packt Publishing Ltd, 2016. ISBN 978-1-78588-604-1.

32 *Services, Load Balancing, and Networking* [online]. [vid. 2019-03-15]. Dostupné z: https://kubernetes.io/docs/concepts/services-networking/

33 HIGHTOWER, Kelsey, Brendan BURNS a Joe BEDA. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. B.m.: O'Reilly Media, 2017. ISBN 978-1-4919-3567-5.

34 LUKSA, Marko. *Kubernetes in Action*. 1 edition. Shelter Island, NY: Manning Publications, 2018. ISBN 978-1-61729-372-6.

35 *Picking the Right Solution* [online]. [vid. 2019-03-15]. Dostupné z: https://kubernetes.io/docs/setup/pick-right-solution/

36 *Concepts* [online]. [vid. 2019-04-13]. Dostupné z: https://kubernetes.io/docs/concepts/

37 DAVIES, Alex. *High Availability MySQL Cookbook*. B.m.: Packt Publishing Ltd, 2010. ISBN 978-1-84719-995-9.

38 *PostgreSQL High Availability Cookbook - Second Edition | PACKT Books* [online]. [vid. 2019-03-16]. Dostupné z: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-high-availability-cookbook-second-edition

39 Introduction. *Vagrant by HashiCorp* [online]. [vid. 2019-04-10]. Dostupné z: https://www.vagrantup.com/intro/index.html

40 How to Install a Kubernetes Docker Cluster on CentOS 7. *HowtoForge* [online]. [vid. 2019-04-15]. Dostupné z: https://www.howtoforge.com/tutorial/centos-kubernetes-docker-cluster/

41 *Net.bridge.bridge-nf-call and sysctl.conf - Libvirt Wiki* [online]. [vid. 2019-04-15]. Dostupné z: https://wiki.libvirt.org/page/Net.bridge.bridge-nf-call_and_sysctl.conf

42 *CoreOS* [online]. [vid. 2019-04-10]. Dostupné z: https://coreos.com/flannel/docs/latest/kubernetes.html

43 *flannel is a network fabric for containers, designed for Kubernetes: coreos/flannel* [online]. Go. B.m.: CoreOS, 2019 [vid. 2019-04-11]. Dostupné z: https://github.com/coreos/flannel

44 CONTAINERUM. Kubernetes networking, easy or not easy? *Containerum* [online]. 12. červenec 2018 [vid. 2019-04-11]. Dostupné z: https://medium.com/containerum/kubernetes-networking-easy-or-not-easy-35a1b7beda51

45 *Weave Net: Network Containers Across Environments | Weaveworks* [online]. [vid. 2019-04-11]. Dostupné z: https://www.weave.works

46 *botpress/server - Docker Hub* [online]. [vid. 2019-04-11]. Dostupné z: https://hub.docker.com/r/botpress/server

47 SERVICES, EMC Education. *Information Storage and Management: Storing, Managing, and Protecting Digital Information in Classic, Virtualized, and Cloud Environments*. B.m.: John Wiley & Sons, 2012. ISBN 978-1-118-23696-3.

48 https://docs.docker.com/engine/reference/builder/. *Docker Documentation* [online]. 9. duben 2019 [vid. 2019-04-11]. Dostupné z: https://docs.docker.com/engine/reference/builder/

49 *Services* [online]. [vid. 2019-04-12]. Dostupné z: https://kubernetes.io/docs/concepts/services-networking/service/

50 TCPDUMP. *Tcpdump/Libpcap public repository* [online]. [vid. 2019-04-12]. Dostupné z: https://www.tcpdump.org

51 *Creating Highly Available Clusters with kubeadm* [online]. [vid. 2019-04-22]. Dostupné z: https://kubernetes.io/docs/setup/independent/high-availability/

52 RAJ, Pethuru, Jeeva S. CHELLADHURAI, Vinod SINGH, Shrikrishna HOLLA, Oskar HANE, Neependra KHARE, Rajdeep DUA, Vaibhav KOHLI, Santosh Kumar KONDURI a Russ MCKENDRICK. *Docker: Creating Structured Containers*. B.m.: Packt Publishing Ltd, 2016. ISBN 978-1-78646-301-2.

53 SMITH, Roderick W. *Linux Administrator Street Smarts: A Real World Guide to Linux Certification Skills*. B.m.: John Wiley & Sons, 2007. ISBN 978-0-470-11674-6.

54 vagrant halt - Command-Line Interface. *Vagrant by HashiCorp* [online]. [vid. 2019-04-18]. Dostupné z: https://www.vagrantup.com/docs/cli/halt.html

55 TLUSTÝ, Jan. *Contribute to honzatlusty/kubernetes-botpress development by creating an account on GitHub* [online]. Shell. 2019 [vid. 2019-04-28]. Dostupné z: https://github.com/honzatlusty/kubernetes-botpress

Přílohy

Příloha A: Skript provison.sh

```
#!/bin/bash
set -e
set -x

useradd kube
mkdir /home/kube/.ssh/
cp /vagrant/id_rsa /home/kube/.ssh/
cat /vagrant/id_rsa.pub > /home/kube/.ssh/authorized_keys
chmod 600 /home/kube/.ssh/id_rsa
chown -R kube:kube /home/kube/
yum install -y vim

ip=$(ip a | grep 'eth1$' | awk '{print $2}' | sed 's?/.*??g')

setenforce 0
sed -i 's?SELINUX=.*?SELINUX=permissive?' /etc/selinux/config

swapoff -a
sed -i '/swap/d' /etc/fstab

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-
ce.repo

cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
        https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

yum clean all
yum install -y vim yum-utils device-mapper-persistent-data lvm2 docker-ce
kubelet kubeadm kubectl

echo "Environment=\"KUBELET_EXTRA_ARGS=--node-ip=${ip}\"" >>
/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
systemctl daemon-reload

for service in docker kubelet; do
  systemctl enable $service
  systemctl start $service
done

cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
```

```
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
EOF

echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
echo "@reboot echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables" >>
/var/spool/cron/root

systemctl restart docker
if hostname | grep -q master; then
  kubeadm init --apiserver-advertise-address=${ip} --pod-network-
cidr=10.244.0.0/16 | grep -A1 '^kubeadm join' > /tmp/join_command
  mkdir -p ~/.kube
  cp /etc/kubernetes/admin.conf ~/.kube/config
  kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl
version | base64 | tr -d '\n')"
  kubectl create deployment botpress --
image=index.docker.io/honzat/kubebot:v11_7_4
  kubectl patch deployment  botpress -p
'{"spec":{"template":{"spec":{"containers":[{"name":"kubebot","env":[{"name":"D
ATABASE","value":"postgres"},{"name":"DATABASE_URL","value":"postgres://botpres
s:botpass@207.154.207.148:5432/botpress"}]}]}}}}'
  kubectl create service nodeport botpress --tcp=3000
  kubectl patch svc botpress -p '{"spec": {"sessionAffinity": "ClientIP"}}'
  kubectl patch svc botpress -p '{"spec": {"ports": [{"port": 3000,"nodePort":
31227,"name": "3000"}]}}'
  kubectl scale --replicas=2 deployment/botpress
else
  sudo -u kube scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
master:/tmp/join_command /tmp
  bash /tmp/join_command
fi
```

Příloha B: Konfigurační soubor pro Vagrant

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# See: https://docs.vagrantup.com/v2/vagrantfile/tips.html


VAGRANTFILE_API_VERSION = "2"

VIRTUAL_MACHINES = {
  :kube01 => {
    :ip             => '10.0.15.21',
    :port           => 3001,
  },
  :kube02 => {
    :ip             => '10.0.15.22',
    :port           => 3002,
  },
  :master => {
    :ip             => '10.0.15.10',
    :port           => 3000,
  }
}

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.hostmanager.enabled = true
  config.vm.box = "centos/7"
  config.ssh.insert_key = false
  VIRTUAL_MACHINES.each do |name,cfg|
    config.vm.define name do |vm_config|
      vm_config.vm.hostname = name
      vm_config.vm.network :private_network, ip: VIRTUAL_MACHINES[name][:ip]
      config.vm.network "forwarded_port", guest: 31227, host:
VIRTUAL_MACHINES[name][:port]
      config.vm.provider :virtualbox do |vb|
        vb.memory = 3072
        vb.cpus = 2
        vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
        vb.customize ["modifyvm", :id, "--ioapic", "on"]
      end # provider
    config.vm.provision "shell",
      path: "provision.sh"
    end
  end
end
```