Automatizace managementu Kubernetes clusterů

Tomáš Turecký

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš Turecký**
Osobní číslo:	**A19118**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Automatizace managementu Kubernetes clusterů**
Téma práce anglicky:	**Automatization of Kubernetes Cluster Management**

Zásady pro vypracování

1. Seznamte se s technologií Docker a Kubernetes.
2. Prostudujte možnosti instalace a nastavení clusteru Kubernetes.
3. Vhodným způsobem vyberte jednotlivé komponenty.
4. Definujte postup jednotlivých kroků pro sestavení a modifikaci clusteru Kubernetes.
5. Vytvořte sadu automatizovaných scripttů pro jednotlivé činnosti.
6. Ověřte výsledky práce ve virtuálním prostředí.

Seznam doporučené literatury:

1. HUANG, Kaizhe a Pranjal JUMDE. Learn Kubernetes Security: Securely orchestrate, scale, and manage your microservices in Kubernetes deployments. Birmingham, UK: Packt Publishing, 2020. ISBN 978-1-83921-650-3.
2. BAIER, Jonathan a Jesse WHITE. Getting Started with Kubernetes: Extend your containerization strategy by orchestrating and managing large-scale container deployments. 3rd edition. Birmingham, UK: Packt Publishing, 2018. ISBN 978-1-78899-472-9.
3. BAIER, Jonathan, Gigi SAYFAN a Jesse WHITE. The Complete Kubernetes Guide: Become an expert in container management with the power of Kubernetes. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1-83864-734-6.
4. BURNS, Brendan. Designing Distributed Systems. O'Reilly Media, 2018. ISBN 978-1-49198-364-5.
5. SAYFAN, GIGI. Mastering Kubernetes. Birmingham, UK: Packt Publishing, 2017. ISBN 978-1-78646-100-1.
6. LUKŠA, Marko. Kubernetes in Action. New York: Manning, 2017. ISBN 9781617293726.

Vedoucí bakalářské práce: **Ing. Peter Janků, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13.7.2024

Tomáš Turecký, v.r.
podpis studenta

ABSTRAKT

Tato práce se zabývá automatickou tvorbou Kubernetes clusteru. V práci jsou probrané možnosti instalace a tvorby clusteru. Vybrané možnosti jsou poté po jednotlivých krocích popsány, včetně příkazů pro jejich docílení. Podle jednotlivých kroků byly vyvinuté scripty pro shell bash, které docílí automatické tvorby Kubernetes clusteru na lokálních strojích. Práce může sloužit jako návod, podle kterého lze manuálně vytvořit Kubernetes cluster v lokálním prostředí, nebo její výsledné scripty jako nástroj, pro tvorbu automatickou. Tato automatická tvorba je určená především pro administrátory lokálních serverů, kteří často potřebují vytvářet nové clustery pro tenanty.

Klíčová slova: kubernetes, automatizace, scriptování, management

ABSTRACT

This thesis deals with the automatic creation of a Kubernetes cluster. The work discusses the possibilities of installing and creating a cluster. The selected options are then described step by step, including commands for achieving them. According to the individual steps, shell bash scripts were developed, which will achieve the automatic creation of a Kubernetes cluster on local machines. The work can serve as a guide, according to which you can manually create a Kubernetes cluster in a local environment, or its resulting scripts as a tool for automatic creation. This automatic creation is primarily intended for local server administrators who often need to create new clusters for tenants.

Keywords: kubernetes, automation, scripting, management

Chtěl bych poděkovat panu doktorovi Petr Janků za jeho podporu a ochotu ve vedení této práce. Dále bych chtěl poděkovat mé blízké rodině za jejich podporu během mého bakalářského studia.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní době je Kubernetes jedná z nejvíce rozšířených a používaných technologií k orchestraci kontejnerů na více strojích. Poskytuje důležitou abstrakci, kdy se může cluster z více serverů chovat jako jeden velký server. Kubernetes clustery lze používat buďto v cloudovém nebo lokálním prostředí.

Manuální tvorba Kubernetes clusteru je zdlouhavá a často repetitivní. Je při ní potřeba vybrat potřebné komponenty clusteru a nainstalovat na každém stroji potřebné nástroje, ještě předtím, než se cluster začne tvořit. Při tvorbě clusteru je poté potřeba mezi nody clusteru předávat informace manuálně, buďto ručním přepisem, nebo vzdáleným kopírováním.

Po vytvoření clusteru je potřeba ještě dalších akcí, jako je tvorba uživatelských účtů, které jsou taky podobně zdlouhavé.

V cloudovém prostředí existuje spousta poskytovatelů, které poskytují automatické vyhrazení zařízení a tvorbu clusterů z nich. V lokálním prostředí ale tyto možnosti chybí. Je více návodů, poskytující informace potřebné k tvorbě lokálního clusteru, ale většina z nich neprobírá možnosti této instalace. Žádné z nich už se nepotýkají automatizací této instalace.

Touto problematikou se zabývá tato práce. Jejím cílem je sepsat potřebné kroky k tvorbě plně funkčního Kubernetes clusteru a pro tyto kroky následně vyvinout scripty, které je budou automatizovat.

Výsledná práce může posloužit jako návod k manuálnímu vytvoření lokálního Kubernetes clusteru použitelný i jednotlivci, kteří se o technologii Kubernetes zabývají poprvé.

Automatické scripty, jejichž tvorba je hlavním cíle práce, mají sloužit jako nástroj pro administrátory lokálních serverů, který nějakým způsobem pronajímají. Když administrátor potřebuje pro tenanta vytvořit cluster, tak místo manuální tvorby by využil scriptů této práce.

Scripty dále mužou posloužit jako nástroje jak pro již zmíněné nováčky s technologií Kubernetes, tak pokročilé uživatele, kteří potřebují vytvořit lokální Kubernetes cluster o více nodech. Vytvořený cluster může sloužit jak pro testování aplikace před přesunem na jiný cluster, tak jako plnohodnotný cluster pro lokální hostování clusterových aplikací.

I. TEORETICKÁ ČÁST

1 DOSAVADNÍ VÝSLEDKY A JINÉ PRÁCE VE STEJNÉ OBLASTI

Během doby existence Kubernetes bylo více snah o automatizaci akcí, o které by se musel starat manuálně administrátor clusteru. Ať už se jedná o automatizaci tvorby clusteru nebo jednotlivé akce, autor této práce našel několik prací, co se tímto tématem zabývají.

Security automation for multi-cluster orchestration in Kubernetes je práce od autorů Daniele Bringhenti, Riccardo Sisto a Fulvio Valenza. Zaměřuje se na automatizaci tvorby síťových nastavení ve více-clusterovém prostředí a podporu mezi-clusterové komunikace. Práce navrhuje a implementuje tzv. *Multi-Cluster Orchestrator,* což je entita nejvyšší úrovně, spravující síťové nastavení všech domén clusterů pod ní založené na požadavcích jejich síťových správců. Práce definuje doménu jako skupinu clusterů pod jednou společností a spravovanou jedním síťovým správcem. Multi-Cluster Orchestrator byl vyvinut v jazyce Java.[1]

Kubernetes with Kubeadm: Fully Automated Installation with Terraform je elektronický článek zabývající se automatizací tvorby Kubernetes clusteru pomocí nástrojů Kubeadm a Terraform v cloudovém prostředí. Článek je ve formě návodu po jednotlivých krocích, obsahující všechny potřebné Terraform konfigurace a příkazy pro kompletní tvorbu clusteru. Využívá cloudového prostředí od Hetzner, containerd jako container runtime a calico jako container network interface.[2]

Cloud-Native Application Validation & Stress Testing through a Framework for Auto-Cluster Deployment je práce od autorů Nikolaos Astyrakakis, Yannis Nikoloudakis, Ioannis Kefaloukos, Charalabos Skianis, Evangelos Pallis a Evangelos K. Markakis. Tato práce se zabývá automatizací ověření cloudově nativních aplikací a automatickou tvorbou Kubernetes clusteru na cloudové platformě OpenStack. Prezentuje framework, který pomocí platformi OpenStack dokáže vytvořit virtuální stroje a z nich vytvořit funkční Kubernetes cluster. [3]

Tato práce se zabývá automatizací tvorbou Kubernetes clusteru. Narozdíl od první zmíněné práce, která se zabývá automatizací zabezpečení pro více již vytvořených clusterů, tato práce je zaměřená na automatizaci tvorby jednoho úplně nového clusteru, bez komplexní automatizace zabezpečení. Dále, zatím co práce druhá a třetí se zabývají automatizací tvorby clusteru, daná automatizace probíhá na cloudovém prostředí. Tato práce se narozdíl od toho zabývá tvorbou clusteru na lokálních strojích. Důležitý rozdíl je také ten, že tato práce automatizuje tvorbu clusteru skrze čisté bash scripty, oproti druhé a třetí práci, které využívají nástrojů Terraform a OpenStack respektivně pro velkou část automatizace.[1][2][3]

2 POUŽITÉ TECHNOLOGIE A NÁSTROJE

2.1 Docker

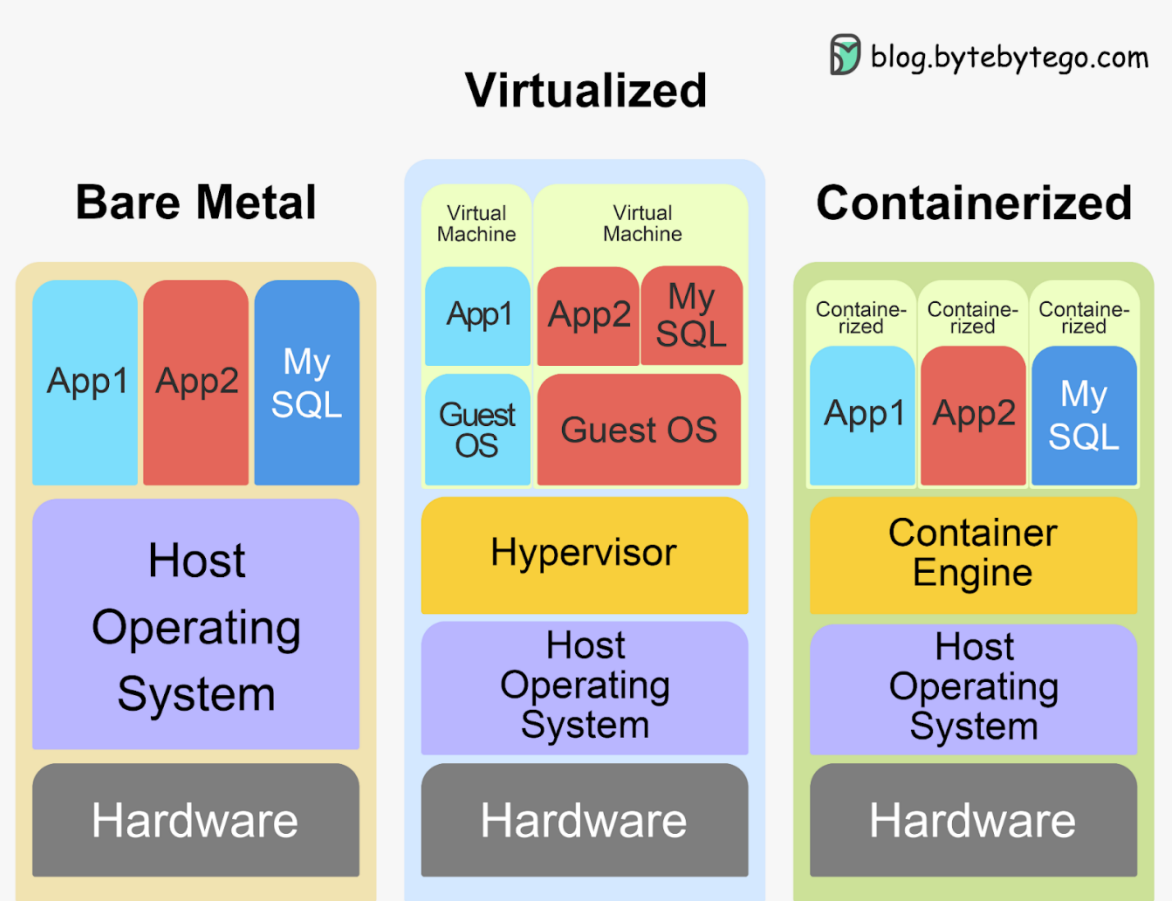
Docker je technologie pro tvorbu, spouštění a sdílení kontejnerových obrazů. Nejedná se o jedinou technologii, ale jednu z nejpopulárnějších. Dříve používaná u Kubernetes jako taková, později její součást containerd, která se stará o spouštění kontejnerů, byla umožněná k použití mimo samostatný Docker. Nyní se používá Docker u Kubernetes hlavně k tvorbě a distribuci kontejnerových obrazů, které se poté spouštějí v Kubernetes clusteru pomocí containerd. Kontejnerové obrazy vytvořené přes Docker lze sdílet přes veřejně dostupné uložiště jako DockerHub, nebo přes vlastní soukromé uložiště.[4][5]

2.2 Kontejnerizace

Kontejnery využívají Linuxových kernel namespaců a cgroup k izolaci procesů od zbytku systému. Každý proces v kontejneru vidí pouze to, co je ve stejném namespacu jako on. To může zahrnovat ostatní procesy, filesystem, síťové rozhraní atd. Pomocí cgroup mají procesy v kontejneru omezené množství systémových zdrojů, které mohou použít. Procesy takto izolované běží na hostitelském operačním systému. Protože běží na hostitelském operačním systému jsou kontejnery oproti virtuálním strojím méně izolované, ale z toho stejného důvodu jsou méně náročné na systém. [4][5][6]

2.3 Virtualizace

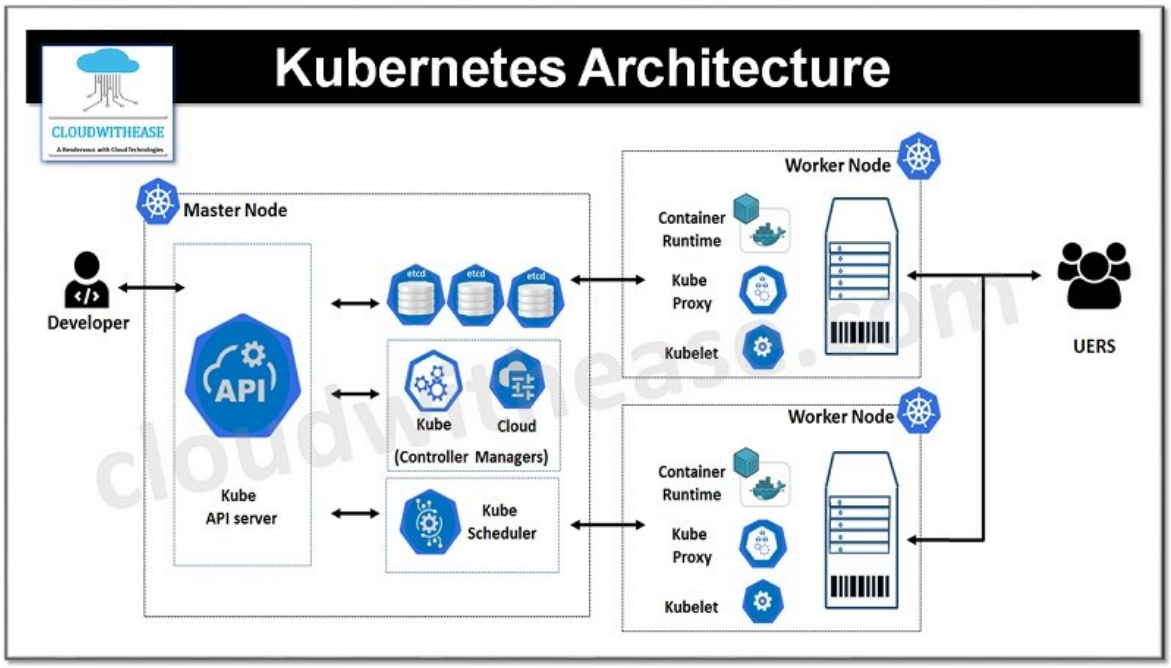
Virtualizace je technologie zabývající se tvorbou virtuálních strojů. Na rozdíl od kontejnerizace se zabývá tvorbou celého virtuálního operačního systému. Oproti kontejnerům poskytují virtuální stroje lepší izolaci a možnost emulace hardwaru. K virtualizaci se vyžívá tzv. *hypervisoru*, který se stará o tvorbu virtuálních stroj a slouží jako abstrakce mezi hardwarem a virtuálním strojem. Virtuální stroje se používají často společně s kontejnery, kde je na fyzickém stroji spuštěn virtuální stroj, ne kterém jsou poté spuštěny samotné kontejnery.[21][22]



Obrázek 1. Rozdíl mezi virtualizací a kontejnerizací [22]

2.4 Kubernetes

Kubernetes je sorfwarový systém, který dovoluje lehké spouštění a spravování kontejnerových aplikací. Kubernetes umožňuje spouštět aplikace na více počítačových nodů, jako by se jednalo o jeden velký počítač. Tvoří abstrakci mezi aplikacemi a infrastrukturou, což ulehčuje jejich vývoj, spouštění a správu. Spouštění aplikací skrz Kubernetes je vždycky stejné, nezávisle na velikosti clusteru. Cluster je ovládán skrz kubectl buďto přes příkazi, nebo (častěji) přes YAML nebo JSON manifesty. Administrátor u hodně akcí neříká clusteru, co má dělat, ale jaký stav clusteru by si přál. Dobrým příkladem je, když administrátor chce zvýšit počet kopií podů ze 2 na 3. Administrátor neřekne clusteru, ať přidá kopii, ale že si přeje, aby byly 3 kopie.[4][7]



Obrázek 2. Architektura Kubernetes clusteru [23]

2.4.1 Node

Nody jsou počítače, co jsou součástí Kubernetes clusteru. Dělí se na master nody (neboli control plane), a worker nody. Na master nodech běží ovládací komponenty, jako api server a controller manager. Při defaultním nastavení pro ně není plánované spouštění aplikačních podů, lze je ale nastavit, aby pro ně plánované bylo. Na worker nodech jsou spouštěné aplikační pody. Komunikace mezi nimi probíhá přes api server bežícím na master nodech. Každý node hostí kubelet, který api server pozoruje, a pokud se na api serveru objeví nějaká změna, co se týká jeho nodu, tak se ji snaží uskutečnit. Nody můžou mít labely (označení), podle kterých lze určit, pro které nody je plánované spuštění podů. V pod manifestu může být určené, aby byl pod plánován pouze pro nody, které mají určitý label. Příklade je, když aplikace v podu potřebuje grafickou kartu. Nody s grafickou kartou budou mít label například gpu=true. V pod manifestu bude pak určené, aby se spouštěl pouze na nodech, které mají label gpu s hodnotou true. [4][7]

2.4.2 Pod

Pody jsou základní jednotkou Kubernetes. Jedná se o nejdůležitější kocept Kubernetes. Všechny ostatní části Kubernetes pody spravují, odhalují, nebo jsou nimi používány. Místo toho, aby se kontejnery spravovaly a spouštěly individuálně, je s nimi zacházeno v podobě podů. Každý pod se skládá z jednoho a více kontejnerů. Všechny kontejnery podu se nachází

na stejném nodu, nemůže se stát, že by jeden kontejner podu byl na jednom nodu a druhý na jiném nodu. Pod může mít jakýkoliv počet kopií (replicas). Díky tomu lze některé části kontejnerizované aplikace lehce horizontálně škálovat. Kubernetes nabízí možnosti automatického horizontálního škálování podle nátěže podů. [4][7]

Proč vůbec používat pody místo jednotlivých kontejnerů? Podle principu kontejnerizovaných aplikací by měl každý kontejner obsahovat jen jeden proces, případně jeho podprocesy. Když je v kontejneru spuštěných více procesů, tak kontejnery ztrácí výhodu izolace a lehkosti logování. Dále, pokud jeden z procesů selže z nějakého důvodu, musí být restartován uvnitř kontejneru místo toho, aby se restartoval celý kontejner. Občas se ale stane, že procesy v těchto kontejnerech musí běžet na stejném nodu, většinou z důvodu komunikace a sdílení souborů. K tomu slouží pody. Pody zajišťují, že se jejich kontejnery nachází na stejném nodu. Dále jim například umožňují sdílet soubory pomocí tak zvaných volumů. [4][7]

2.4.3 ReplicaSet

Administrátor může vytvářet pody a jejich kopie ručně a po jednom. Použije stejný YAML manifest, kde bude jenom měnit jméno podu. V případě, že by administrátor chtěl počet podů snížit, může jednotlivé pody manuálně odstranit. Takový postup je ale nejen repetivní, jde i proti principu Kubernetes, kde administrátor má cluster žádát o určitý stav clusteru, místo toho aby ho sám nastavoval. Dále, pokud z nějakého důvodu pod selže, administrátor by ho musel restartovat manuálně. Zde hrají důležitou roli ReplicationControllery a ReplicaSety. Dříve existoval pouze ReplicationController. ReplicaSet je nová generace ReplicationControlleru, používaný hlavně společně s Deploymentem. ReplicationController a ReplicaSet fungují velice podobně. V YAML manifestu pro jejich tvorbu jsou nejen vlastnosti samotného ReplicationControlleru nebo ReplicaSetu, ale i template, podle které budou vytvářet pody co následně spravují. Starají se nejen o tvorbu svých podů, ale i o jejich restartování a případné odstranění. Které pody spravují je určeno pomocí labelů. Tyto labely fungují stejně jako u již zmíněných nodů, kde každý label má klíč a hodnotu.[4][7]

Hlavním rozdílem mezi ReplicationControllerem a ReplicaSetem je způsob výběru podů, které spravují, a způsob průběžné aktualizace (rolling update). ReplicationController používá selector podporující pouze rovnost, například prostředí=produkce, kde ReplicationController spravuje pody, jejichž label je prostředí=produkce. ReplicaSet oproti tomu používá selector podporující jak rovnost, tak sety, například prostředí in (produkce, testování), kde ReplicaSet spravuje jak pody, jejichž label je prostředí=produkce, tak pody, jejichž label je

prostředí=testování. Jak ReplicationController tak ReplicaSet podporují tzv. rolling update. Rolling-update pro ReplicationController a rollout pro ReplicaSet. Při rolling updatu jsou pody postupně vytvářené a mazány tak, aby byl vždy určitý počet podů dostupný podle nastavení administrátora. Rolling update se používá v situaci, kdy administrátor chce aktualizovat pody během běžné činnosti clusteru, která zabraňuje možnosti všechny pody a jejich ReplicationController či ReplicaSet smazat. Rozdíl mezi rolling-update a rollout je ten, že rolling-update běží na kubectl, zatím co rollout běží na master nodu. Od rolling-update se již odpustilo z důvodu jeho spoléhání se na spojení mezi klientem a master nodem. Protože rolling-update běží na kubectl klientského počítače, pokud dojde k selhání spojení, rolling-update zůstane nedokončený. Narozdíl od rolling-update, rollout běží na master nodu. Pokud dojde k přerušení signálu mezi klientem a master nodem, rollout stále pokračuje na master nodu. Důležité je ale zmíňit, že rollout je možný použít pouze, pokud je ReplicaSet součástí Deploymentu. [4][7]

2.4.4 Deployment

Pody v Kubernetes clusteru lze vytvořit buďto jednotlivě, nebo pomocí ReplicaSetu. Existuje ale ještě vyšší koncept, tzv. Deployment. Deployment slouží k nasazování aplikací a jejich aktualizací deklaračně, místo využití samotných ReplicaSetů k manuální aktualizaci podů. Když je Deployment vytvořený, jsou pod ním vytvořeny i potřebné ReplicaSety, které tyto pody spravují. Když je kontejner obraz Deploymenty aktualizovaný, je pod ním vytvořen nový ReplicaSet. Deployment poté postupně snižuje počet kopií u původního ReplicaSetu a zvyšuje počet u nového. Výsledek je 1 ReplicaSet s novými kontejner obrazy.[4]

2.4.5 Service

Pody clusteru většinou potřebují mezi sebou komunikovat. Používat ip adresy jednotlivých podů je ale velice nepraktické, protože většinou existuje více identických podů a jsou často mazány a znovu vytvářeny, kdy pokaždé dostanou jinou novou ip adresu. Zde přichází na řadu Servisy. Service poskytuje pevný bod, skrz který lze komunikovat s pody za ním se schovávajícími. Service má pevně danou ip adresu a porty po dobu jeho existence. Clienti s ním můžou komunikovat, a jejich komunikace je pak přesměrována na jeden z podů. Komunikace může tedy probíhat i přes to, že se ip adresy podů neustále mění.[4]

2.4.6 Namespace

Namespace je API objekt sloužící k rozdělení clusteru. U většiny API objektů lze při tvorbě určit, ve kterém Namespace mají být vytvořené. API objekty můžou mít stejný název, jako objekt v jiném Namespace, oproti objektech ve stejném, se kterým název sdílet nemůžou. V defaultním nastavení neposkytují izolaci mezi objekty v různých Namespace. Co ale poskytují, je možnost určení uživatelům pravomoci v nich. Uživatel může mít například pravomoci pro tvorbu Podů v namespace *ns1*. Tento uživatel nebude schopen jakýchkoliv jiných akcí jak v tomto Namespace, tak Namespace jiných. [4]

2.5 Alternativní technologie pro orchestraci kontejnerů

Kubernetes je jedna z nejpopulárnějších možností pro orchestraci clusterů a jejich aplikací. Nejedná se ovšem o jedinou. Existují alternativy pro uživatele, kteří potřebují jednodušší nebo jinou funkcionalitu. Nejvíce používané alternativy jsou Docker Swarm a Nomad.

2.5.1 Docker Swarm

Vestavěná součást nástroje Docker. Slouží k orchestraci a škálování kontejnerů v clusteru strojů. Je používán deklarativně stejně jako Kubernetes. Je vhodný pro hostování škálovatelných aplikací s využitím pouze standartní Docker instalace.[25]

2.5.2 Nomad

Orchestrátor od společnosti HashiCorp. Oproti Kubernetes je jednodušší, se zaměřením čistě na orchestraci clusteru a plánování. Díky jednoduchosti podporuje cluster s počtem nodů 10000 a více, oproti Kubernetes, které podporuje do 5000. Dále, zatím co Kubernets je specializován pro Linux kontejnery, Nomad je obecnější a podporuje například Internet Information Services pro Windows, nebo Qemu.[26]

2.6 Debian

Debian je open-source operační systém, přesněji distribuce GNU/Linuxu. Jedná se o jednu z nejvíce používaných distribucí. Distribuce je především známá svojí stabilitou v oblasti aktualizací. Nové verze bývají široce a dlouhodobě testovány, než jsou vydány ve stabilní verzi.[15]

2.7 OpenSSH

Nástroj využívající Secure Shell protokolu. Jedná se o jednu z nejvíce rozšířených implenetací tohoto protokolu, často k dispozici v defaultních instalacích ruzných Linux distribucí, včetně v této práci použité distribuce Debian. To, a velmi jednoduchý způsob použití, je důvod proč je OpenSSH použito v této práci oproti možným jiným alternativním nástrojům implementující SSH.[13]

2.7.1 SSH

Secure Shell protokol, zkráceně SSH, je protokol pro tvorbu zašifrovaných komunikačních kanálu mezi dvěma zařízeními, kde na jednom zařízení se nachází ssh server a na druhém ssh klient. [13]

2.7.2 SSH klíče

Základní způsob, jak se autentizovat skrz SSH připojení je pomocí uživatelského jména a hesla pro uživatele na zařízení, na kterém se SSH server nachází. Toto je přijatelný způsob, pokud se jedná o občasnou manuální práci přes SSH, případně o manuální spouštění scriptu přes SSH pro jedno zařízení. Tento přístup ovšem pokulhává, když se jedná o použití časté, nebo větší automatizaci scriptů. Ve velmi podobné situaci se nachází scripty této práce. Zde přichází na řadu SSH klíče. SSH podporuje použití public-private klíčových párů pro autentizaci připojení namísto klasického jména a hesla. [13]

2.8 Bash

Bash je shell, také známo jako command language interpreter(CLI), pro operační systém GNU. Je vysoce kompatibilní s momentálním Unix shellem *sh* a zabudovává užitečné funkcionality z Korn shellu *ksh* a C shellu *csh.* Operační systém GNU poskytuje i jiné shelly, ale Bash je shellem defaultním. Oproti shellu sh poskytuje bash vylepšenou funkcionalitu ze strany interakce a programování. V této práci je bash používaný k programování všech scriptů. To že se jedná o defaultní shell GNU a jeho vylepšená funkcionalita v oblasti programování oproti sh jsou důvody, proč je použit v této práci. [14]

2.9 Iptables

Iptables je linuxový nástroj pro tvorbu a spravování firewallových packetových pravidel. Dřív se jednalo o nejrozšířenější nástroj k tomuto účelu. V dnešní době bývá ale nahrazován

nástupcovým nástrojem Nftables, který funguje velice podobně ale s rozšířenou funkcionalitou. V nejnovější verzi distribuce Debian bylo iptables také nahrazenou nftables, Kubernetes ale nftables zatím nepodporuje, proto je potřeba instalace iptables.[16]

3 MOŽNOSTI INSTALACE A NASTAVENÍ CLUSTERU

3.1 Možnosti instalace

Existuje několik typů způsobů, jak vytvořit funkční Kubernetes cluster. Způsoby instalace lze rozdělit na jedno-nodové, nespravované mnoho-nodové, spravované mnoho-nodové a manuálně instalované. Jedno-nodové clustery jsou primárně určené pro použití jako vývojáský sandbox, využitý k testování nebo učení. Nespravované mnoho-nodové jsou clustery námi instalované a spravované, buďto lokálně nebo na cloudu. Spravované mnoho-nodové jsou většinou cloudové, spravované třetí stranou, většinou poskytovatelem cloudové platformy. Při manuální instalaci se o každý krok instalace stará administrátor.

3.1.1 Jedno-nodové

Primárně určené k vývoji a testování, jedno-nodové způsoby instalace jsou velice jednoduché. Mezi tyto způsobi patří Kubernetes přes Docker Desktop a Minikube. Tyto clustery se skládají pouze z jednotoho nodu, který je zároveň master i worker node. Všechny pody běží na tomto nodu. Kvůli omezení na jeden node je nelze použít v produkčním prostředí.[7]

Kubernetes přes Docker Desktop

Určené hlavně pro uživatele Docker Desktop. Pro použití Kubernetes přes Docker Desktop stačí povolit Kubernetes v nastavení Docker Dashboard. Kubernetes takhle spuštěné nelze nijak konfigurovat a běží jako Docker kontejner na lokálním systému.[5][6]

Minikube

Minikube je Kubernetes distribuce zaměřená na lehkost instalace a práce, určená hlavně k testování a studování využití Kubernetes. K použití stačí pouze instalace Docker Enginu a Minikube. Poté stačí použít jeden příkaz ke spuštění clusteru. Minikube je vyvýjeno vývojáři Kubernetes. [4][8] [10]

3.1.2 Nespravované mnoho-nodové

Plněhodnotné clustery skládající se z jednoho nebo více master nodů a několika worker nodů. Jedná se hlavně o clustery nainstalované pomocí instalačních nástrojů. Mezi tyto nástroje patří kubeadm, kops, nebo kubespray. Tyto clustery lze tvořit jak lokálně, tak na cloudu. Jedná se o clustery vhodné pro produkční prostředí. [8]

Kubeadm

Oficiální nástroj pro tvorbu Kubernetes clusterů, od vývojářů Kubernetes. Myšlenka za projektem je taková, že kubeadm bude základ tvorby všech Kubernetes clusterů, zatím co pokročilé a na míru tvořené nátroje budou vytvořené na kubeadm. Z tohoto důvodu je kubeadm vybráno jako způsob tvorby clusteru.[4][8][11]

Kops

Jedná se o nástroj pro tvorbu Kubernetes clusterů na určitých cloudových platformách. Oficiálně podporuje platformy Amazone Web Services a Google Cloud Platform. Nástroj se stará jak o tvorbu a správu clusteru, ale také se postará o nastavení potřebné cloudové infrastruktury. [8][12]

Kubespray

Kubespray kombinuje Kubernetes a Ansible, dovolující sestavovat Kubernetes cluster pomocí Ansible playbook. Podporuje jak lokální, tak cloudové clustery. Snaží se držet balanc mezi rychlostí a flexibilitou. [8]

3.1.3 Spravované mnoho-nodové

Clustery instalované hlavně na cloudových platformách. Poskytovatel cloudové platformy se postará o instalaci i spravování clusteru. Tvorba clusteru na cloudu pomocí servisů poskytovatele bývá většinou velmi jednoduchá, oproti instalaci lokální. Vhodné pro produkční prostředí, omezené hlavně cenou samotné cloudové platformy. Mezi tyto servisi patří Elastic Kubernetes Service určené pro Amazon Web Services, Azure Kubernetes Service určené pro Microsoft Azure, Google Kubernetes Engine určené pro Google Cloud Platform a Oracle Kubernetes Engine určené pro Oracle Cloud Infrastructure. [8]

3.1.4 Instalované manuálně

U všech dosavadně zmíněných způsobů instalace Kubernetes clusteru hrála roli jistá úroveň automatizace. Ať už vetší (u jedno-nodových a spravovaných mnoho-nodových) nebo menší (nespravované mnoho-nodové), je při tom využit nějaký nástroj, který se postará o instalaci základních komponentů. Existuje ale ještě kategorie způsobů, kde se musí administrátor postarat o všechno sám, ať už na lokální nebo cloudové platformě. [8]

Kubernetes The Hard Way

Při diskusi o manuláních instalacích je určitě potřeba zmíňit Kubernetes The Hard Way (Kubernetes Těžkou Cestou), návod vytvořený pod vedením Kelseyho Hightowera. Návod je optimalizový pro vzdělání, kde se čtenář naučí každou jednotlivou činnost potřebnou k tvorbě funkčního Kubernetes clusteru. Je určený pro pracovníky v oboru, kteří mají v plánu podporovat produkční Kubernetes cluster a potřebují vědět, jak všechny jeho části pasují do sebe. Originální návod je zaměřený na tvorbu clusteru na Google Cloud Platformě, ale existují publikace probírající, jak návod adoptovat pro lokální clustery. [8]

3.2 Jeden nebo více master nodů

Kubernetes clusteru stačí k funkčnosti jeden master node, skrz který se cluster inicializoval. Dvě z důležitých vlastností Kubernetes clusteru jsou ale redundance a horizontální škálovatelnost, ať už se jedná o tvorbu více identických podů, které dovolují funkčnost aplikace, když jsou některé pody nedostupné či zaneprázdněné, nebo více worker nodů, což umožňuje funkci clusteru i když některé worker nody nejsou zrovna k dispozici. Ve stejném duchu podporuje Kubernetes existenci více master nodů v jednom clusteru. Více master nodů v clusteru umožňuje funkci Kubernetes clusteru, i přes to že nejsou některé master nody k dispozici. V případě že je rozhodnuto pro více master nodů je ale potřeba lichý počet, aby aby v případě nesrovnalosti informací na nodech existovala většina, která má identické informace. [4]

3.3 Tvorba podů na master nodech

Když uživatel vytvoří na clusteru pod, ať už manuálně nebo skrz Deployment, bude Kubernetes Schedulerem přiřazený na jeden z dostupných a vhodných nodů. Při defaultním nastavení jsou tyto pody tvořené na vhodných worker nodech. Na master nodech se tyto pody vůbec nespouštějí. To je z důvodu, že jsou master nody určené především ke spravování clusteru a je důležité, aby jejich zdroje nebyly zahlcené běžnými aplikačními pody. V nejhorší situaci by mohly tyto pody způsobit i crash tohoto master nodu, což je v případě clusteru s jedním master nodem katastrofální, a v případě clusteru s více master nody značně omezující.

Cluster vymezuje, na kterých nodech se můžou pody spouštět, tzv. *taintem*. V případě master nodů se jedná specificky o taint s efektem *NoScheduling*. Tento taint zabrání přiřazení podů, které nemají specifickou toleranci, na dané nody. Existují tedy 2 způsoby, jak přimět

Scheduler, aby přiřadil pod na master node. První z nich je určit podu toleranci, která to dovolí. Druhý způsob, ten probírán v této práci dále, je změna taintu samotného nodu, v tomto případě jeho odstranění.[4]

3.4 Uživatelské účty

Důležitou administrátorskou činností je správa uživatelských účtů na clusteru. V Kubernetes ale, na rozdíl od účtů pro servisy aplikací, neexistuje objekt, který by sloužil pro tvorbu uživatelského účtu. Místo toho se uživatelé autentizují předložením certifikátem podepsaným clusterovou centrální autoritou.

Při tvorbě Kubernetes clusteru je vytvořený defaultní administrátorský účet. Tento účet umožňuje spravování všech částí clusteru. Tento účet ale není vhodný pro použití pro většinu uživatelů clusteru. Pro uživatele, kromě hlavní administrátora, je důležité vytvořit účty, které mají omezené pravomoci. V praxi to vypadá tak, že každý uživatel má určené namespacy, ve kterých má přístup k různým akcím a různým API objektům. Příkladem by bylo, že uživatel *user1* má v namespace *dev* pravomoci pouze pro spravování deploymentů a zobrazování serviců. [4]

3.4.1 RBAC API Objekty

Podepsáním certifikátu clusterem je uživatel vytvořen. Stále je ale potřeba určit jeho pravomoci. K tomu slouží API Objekty *Role*, *ClusterRole*, *RoleBinding a ClusterRoleBinding.* Jsou součástí tzv. *Role-base access control* (neboli *RBAC*), což je způsob regulace přístupu založený na rolích individuálních uživatelů. Jedná se v nejnovějších verzí Kubernetes o defaultní metodu autorizace. Role a ClusterRole obsahují pravomoci dané roli. Role je určená pro specifický namespace, zatím co ClusterRole se vztahuje na celý cluster. RoleBinding a ClusterBinding propojuje Role a ClusterRole se specifickými uživateli nebo listem uživatelů. Stejně jako u Role a ClusterRole, RoleBinding se vztahuje na specifický namespace, zatím co ClusterRoleBinding na celý cluster. [4]

II. PRAKTICKÁ ČÁST

4 VÝBĚR JEDNOTLIVÝCH KOMPONENTŮ

Kubernetes cluster využívá různých komponentů k jeho funkčnosti. Komponenty lze rozděli na potřebné a dobrovolné. Potřebné komponenty jsou ty, bez kterých cluster nemůže fungovat. Některé z potřebných komponentů jsou volitelné, kde jsou sice potřebné k funkci clusteru, ale nejsou dodané instalačním nástrojem jako například kubeadm. Mezi tyto potřebné volitelné patří container runtime a pod network addon.

4.1 Container runtime

Důležitou součástí Kubernetes jsou kontejnery. Je na nich založený celý princip Kubernetes. Kontejnery jsou tvořený a běží skrz tzv. Container Runtime. Existuje několik možností, které jsou běžně využívané u Kubernetes.

containerd

Dřívě používaný pouze pod Dockerem pod názvem dockerd. Nyní pod Open Container Initiative. Jeden z nejvíce používaných container runtime interfaců. Jako dřívější součást Dockeru dokáže spravovat uložiště a přesouvání kontejner obrazů. Využívá runC container runtimu.[18]

CRI-O

Vyvíjené původně pod Open Container Initiative jako alternativa pro Docker Engine. Umožňuje použití více container runtimů, jako je runC nebo Kata. [18]

cri-dockerd (Docker Engine)

Container runtime interface poskytující možnost využití Docker Engine jako container runtime s Kubernetes. Dovoluje práci s Kubernetes clusterem skrz Docker API. [19]

Mirantis Container Runtime

Komerčně dostupná verze cri-dockerd od Mirantis. Díky tomu má větší poskytnutou podporu od vývojářů. [19][20]

Zvolený container runtime

V této práci bylo rozhodnut pro použití containerd jako container runtime interface. Jedná se díky svému původu jako dockerd o jeden z nejvíce vyvinutých container runtime interfaců. Ze stejného důvodu je vhodný, pokud je Docker použit na tvorbu používaných obrazů.

4.2 Network addon

Dalším velice důležitým komponentem clusteru jsou Network Addony. V defaultní instalaci Kubernetes neexistuje způsob, jakým můžou pody mezi sebou komunikovat. Je očekávané, že administrátor poskytne addon podle jeho potřeb. Protože network addony běží v podobě podů, je jejich instalace většinou jednoduchá a z velké části podobná.

Flannel

Je zaměřený čistě na poskytování sítě. Hlubší nastavení sítě clusteru Flannel nepodporuje. Díky tomu je známý svojí jednoduchostí instalace a nastavení, zaměřený na robustnost a kompatibilitu. Využívá překryvné sítě IPv4 na úrovní 3. [17]

Calico

Na rozdíl od addonu Flannel, Calico podporuje jak poskytování sítě, tak její nastavení. Oproti Flannelu je komplikovanější k použití. Má ale více funkcionality. Na rozdíl od Flannelu nepoužívá překryvnou síť, ale BGP protokol. [17]

Canal

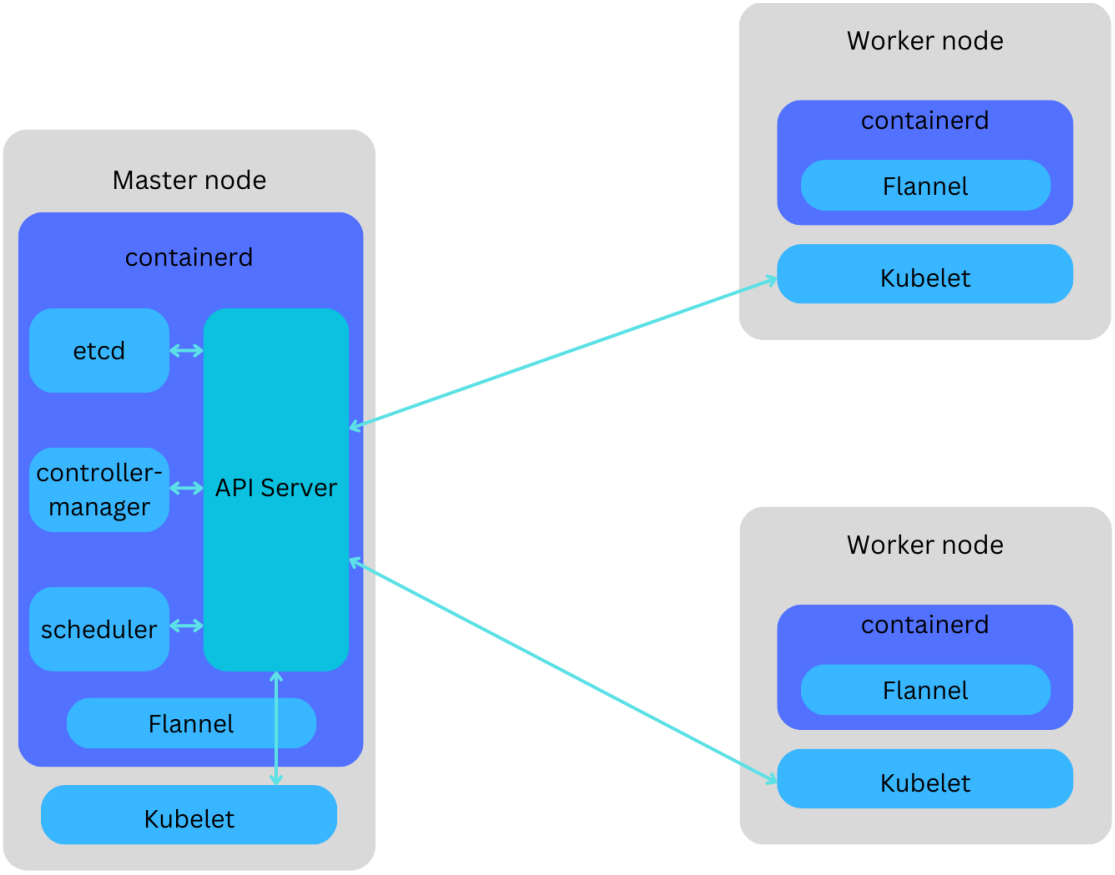
Dříve addon jako takový, v dnešní době se od samostatné implementace upustilo a pouze podporuje kombinaci addonů Calico a Flannel. Flannel je využít pro samotnou síť, zatím co Calico je využito pro její nastavení. [17]

Weave Net

Tento addon poskytuje jak síť, tak možnosti síťového nastavení. Na rozdíl od ostatních poskytuje jednoduché šifrování sítě. Jeho síť využívá routerových komponentů vytvořených na každém nodu. [17]

Zvolený network addon

Jako zvolený network addon by vybrán flannel. Poskytuje základní potřebnou funkcionalitu. Neumožňuje sice tvorbu síťových pravidel, ale to je jednoduché doinstalovat ostatními network addony podle potřeby uživatele.



Obrázek 3. Cluster s vybranými komponenty

5 POSTUP SESTAVENÍ A MODIFIKACE CLUSTERU

Existuje několik způsobů jak sestavit Kubernetes cluster. V této práci autor popisuje způsob sestavení clusteru pomocí nástroje kubeadm. Kubeadm projekt je pod projektem Kubernetes a jedná se o oficiální nástroj k instalaci Kubernetes clusterů.

5.1 Příprava nodů před incializací clusteru

Instalace na obou typech nodů je podobná. Oba typy potřebují mít nainstalované kubeadm, kubelet a containerd. Požadavky na všechny nody jsou následující: 2GB nebo víc, 2 procesorové jádra nebo více, unikátní hostname, unikátní MAC adresu, unikátní product_uuid, některé porty musí být otevřené a musí mít vypnutý SWAP. Všechny nody také musí být na stejné síti. Hostname, MAC adresa a product_uuid by měl být problém pouze u virtuálních strojů které byly naklonované. Instalace je popsaná pro čistou instalaci operačního systému Debian 12.5. Instalace probíhá pod uživatelem root. Instalace je samozřejmě možná i přes jiný účet, který má přístup k příkazu *sudo*.

5.1.1 Instalace na master nodu

Potřebné porty

Master node, nebo-li control-plane, potřebuje otevřené porty pro tyto servisi: 6443 pro Kubernetes API server, 2379-2380 pro etcd server client API, 10250 pro Kubelet API, 10259 pro kube-scheduler a 10257 pro kube-controller-manager. Všechny otevřené pro vstupní připojení a TCP protokol. To lze udělat přes využívaný firewall. V případě systému Debian použitého v této práci toho jde docílit pomocí nástroje iptables, jenž byl dříve defautní firewall této distribuce. Nyní byl nahrazen nástrojem nftables. Proto je iptables potřeba nainstalovat. To lze příkazy:

```
apt-get install iptables
```

V defaultním nastavení se ale distribuce Debian se pouze tváří, že využívá iptables. Ve skutečnosti využívá nftables na pozadí. Z tohoto důvod je potřeba změnit iptables z nft na legacy. Změnu lze provést příkazy:

```
update-alternatives iptables /usr/sbin/iptables-legacy
update-alternatives ip6tables /usr/sbin/ip6tables-legacy
```

Otevření portu je možné vytvořením pravidel. Pro tvorba pravidla je potřeba určit směr cestování packetu, protokol packetu, port, pro který je určený, a akci co s packetem dělat.

Příkaz má formát:

```
iptables -A <směr> -p <protokol> --dport <port> -j <akce>
```

Takže pro socket 6443 se jedná o příkaz:

```
iptables -A INPUT -p tcp --dport 6443 -j ACCEPT
```

Tímto příkazem je potřeba přidat pravidla pro všechny porty popsané na začátku kapitoly. Tyto příkazy ale vytvoří pravidla pouze pro aktuální session operačního systému, po jeho restartu budou smazány. To je ale samozřejmě nevhodné. Tomu lze předejít buďto konfigurací, která se bude spouště po každém zapnutí počítače, nebo pomocí užitečného nástroje *iptables-presistent*. Tento nástroj automaticky vytvoří všechna pravidla uložená v souboru */etc/iptables/rules.v4*, který si vytvořil při instalaci. Nástroj lze instalovat příkazem:

```
apt-get install iptables-presistent
```

Iptables podporuje vypsání všech pravidel v úložném formátu za pomoci příkazu *iptables-save*. Pomocí tohoto příkazu lze pravidla tedy uložit do souboru */etc/iptables/rules.v4*.

```
iptables-save > /etc/iptables/rules.v4
```

Instalace Container Runtimu

Dále je potřeba nainstalovat container runtime. V tomto případě containerd. Před nainstalováním samotného container runtimu je potřeba použít následující příkazy pro načtení potřebných kernelových modulů. Příkazy také slouží ke změně konfigurací a jejich načtení.

Příkazy k načtení potřebných modulů a konfigurace:

```
cat <<EOF | tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
modprobe overlay
modprobe br_netfilter
```

Příkazy ke změně konfigurace sysctl a načtení této konfigurace:

```
cat <<EOF | tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF
sysctl --system
```

Ověřit že jsou moduly načteny lze příkazy:

```
lsmod | grep br_netfilter
lsmod | grep overlay
```

Ověřit že jsou proměnné v konfiguraci sysctl správně nastavené na 1 lze následujícím příkazem:

```
sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
```

Nyní lze nainstalovat předtím zmíněný container runtime containerd. Potřebné binární soubory containerd a binární soubory které potřebuje, jako runc a cni-plugins lze získat z repozitáře na githubu. Soubory lze stáhnout manuálně nebo pomocí nástrojů jako wget. Příkaz pro stáhnutí nejaktuálnějších potřebných souborů v době psaní:

```
wget
https://github.com/containerd/containerd/releases/download/v1.6.15/containerd-1.6.15-linux-amd64.tar.gz
&& wget https://raw.githubusercontent.com/containerd/containerd/main/containerd.service
&& wget
https://github.com/opencontainers/runc/releases/download/v1.1.4/runc.amd64
&& wget
https://github.com/containernetworking/plugins/releases/download/v1.2.0/cni-plugins-linux-amd64-v1.2.0.tgz
```

Tímto příkazem se potřebné soubory stáhnou do adresáře ve kterém se uživatel nachází. Pro instalaci containerd stačí tar archív extrahovat do adresáře /usr/local.

```
tar Cxzvf /usr/local containerd-1.6.15-linux-amd64.tar.gz
```

Protože cluster bude používat systemd jako cgroup manager, musí systemd také spouštět containerd. K tomu je potřeba přesunout soubor containerd.service do adresáře /etc/systemd/system.

```
mv containerd.service /etc/systemd/system/containerd.service
```

Poté lze containerd spustit následujícími příkazy:

```
systemctl daemon-reload
systemctl enable --now containerd
```

Následně je potřeba nainstalovat runc do lokace /usr/local/sbin/runc.

```
install -m 755 runc.amd64 /usr/local/sbin/runc
```

Pro instalaci cni-plugins je potřeba vytvořit adresář v lokaci /opt/cni/bin a poté do něj tar archív extrahovat.

```
mkdir -p /opt/cni/bin
tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.1.1.tgz
```

Nakonec je potřeba vygenerovat defaultní konfigurační soubor. Ten se generuje do vytvořeného adresáře.

```
mkdir /etc/containerd
containerd config default > /etc/containerd/config.toml
```

Po vytvoření konfiguračního souboru je potřeba ho upravit, aby containerd využíval systemd jako cgroup manager. V souboru je potřeba pod:

```
[plugins."io.containerd.grpc.v1.cri".containerd.runti-
mes.runc.options]
```

Změnit hodnotu SystemdCgroup na true. Po této změně je potřeba restartovat containerd příkazem:

```
systemctl restart containerd
```

Restartováním containerd je dokončená instalace container runtimu.

Instalace kubeadm, kubelet a kubectl

Následuje instalace kubeadm, kubeletu a kubectl. Instalace probíhá přes apt. Nejprve je potřeba zkontrolovat, že apt má nejaktuálnější index. Dále se stáhnout potřebné nástroje: apt-transport-https, ca-certificates a curl.

```
apt-get update
apt-get install -y apt-transport-https ca-certificates curl
```

Dále je potřeba stáhnou veřejný podpisový klíč z Kubernetes repozitáře. Klíč se stáhne do nově vytvořené složky. Lze ho stáhnout pomocí nově staženého nástroje curl.

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key  | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
```

Následně je potřeba přidat samotný Kubernetes repozitář do seznamu zdrojů apt. Jako ověřovací klíč nově přidanému repozitáři je potřeba určit nově stažený klíč.

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /" | tee /etc/apt/sources.list.d/kubernetes.list
```

Poté se apt index znovu aktualizuje, aby index obsahoval nově přidaný repozitář. Z nově přidaného repozitáře se následně stáhnou nástroje kubelet, kubeadm a kubectl. Dále se zakáže těmto staženým nástrojům aktualizace. U těchto tří nástrojů je důležité, aby byly na stejné verzi. Kubeadm nepodporuje automatickou aktualizaci těchto nástrojů, takže při automatické aktualizaci je velice pravděpodobné, že přestanou s existujícím clusterem fungovat. To je důvod k zakázání automatických aktualizací.

```
apt-get update
apt-get install -y kubelet kubeadm kubectl
apt-mark hold kubelet kubeadm kubectl
```

Dále je potřeba zakázat SWAP. Kubernetes, specificky kubelet, SWAP nepodporuje.

```
swapoff –a && sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Po nainstalování kubeadm lze master node inicializovat. Inicializační příkaz se postará o spuštění všech nutně potřebných komponentů. Když není kubeadm nebo jiný instalační nástroj použití, je potřeba tyto komponenty, jako plánovač (kube-scheduler) a správce ovladačů (controller-manager), instalovat a spouštět manuálně. Tyto komponenty jsou spouštěny v containarech v podobě podů a součástí namespace kube-system.

5.1.2 Instalace na worker nodu

Instalace na worker nodu probíhá velice podobný způsobem jako na master nodu. Je zde pár odlišností, jako nepotřeba instalace kubectl a potřeba otevření jiných portů. Protože na

worker nodu neběží ovládací komponenty, není pro ně nutné otevírat porty. Jediný komponent, který potřebuje otevřený port na worker nodu, je Kubelet API a to port 10250. Dále je potřeba otevřít rozsah portů 30000-32767 pro NodePort servisy, což u master nodu potřeba nebylo. Zatím co port 10250 lze otevřít příkazem již použitým v předešlé kapitole, a to příkazem:

```
iptables -A INPUT -p tcp --dport 10250 -j ACCEPT
```

Pro rozsah portů 30000-32767 je potřeba použít příkaz jiný. Je samozřejmě možné pro ně tvořit pravidla port po portu, kde by se vytvořilo 2768 různých pravidel. Mnohem snažší ale je využít parametru -m multiport.

```
iptables -A INPUT -p tcp -m multiport --dports 30000:32767 -j ACCEPT
```

Tímto příkazem se vytvoří pravidlo pro celý rozsah. Dále je potřeba nainstalovat containerd container runtime. Instalace probíhá identickým způsobem jako v předešlé kapitole. Načtení potřebných modulů, změna konfigurace sysctl, stáhnutí souborů z githubu a jejich extrahování, tvorba a změna containerd konfiguračního souboru. Zase stejným způsobem jako v předešlé kapitole se poté stáhne podpisový klíč, přidá se repozitář do apt a nainstaluje se kubelet a kubeadm. Protože se instaloval kubelet, je také potřeba zakázat SWAP. Kubectl není potřeba instalovat.

5.2 Inicializace clusteru a připojování nodů

5.2.1 Inicializace clusteru přes kubeadm

Pro inicializaci stačí jeden kubeadm příkaz. Tento příkaz se musí provést na jednom z master nodů.

```
kubeadm init [flags]
```

Tento příkaz může využít konfiguračního souboru, kde lze specifikovat vlastnosti, které potřebujeme, jak pro samotný cluster, tak pro kubelet. Jelikož je systemd využiván jako cgroup správce, je potřeba aby si toho byl kubelet vědom. Dále, jelikož je flannel použit pro spravování pod sítě, je potřeba specifikovat rozsah pod adres (podCIDR), které má flannel přidělovat podům. Obě tyto vlastnosti lze nastavit přes zmíněný konfigurační soubor. Tento konfigurační soubor se poté předá jako parametr kubeadm init příkazu. Konfigurační soubor se skládá z dvou částí, jedna pro vlastnosti clusteru a druhá pro vlastnosti kubeletů. Soubor je v podobě YAML manifestu.

```
kind: ClusterConfiguration
apiVersion: kubeadm.k8s.io/v1beta3
networking:
 podSubnet: "10.244.0.0/16"
---
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
cgroupDriver: systemd
```

Pokud je konfigurační soubor uložen například pod názvem kubeadm-config.yaml, tak inicializační příkaz, který ho použije, je následující:

```
kubeadm init --config <konfigurační soubor>
```

Inicializací clusteru na master nodu na něm vytvořený defaultní administrátorský konfigurační soubor pro kubectl na lokaci */etc/kubernetes/admin.conf*. Pro jeho použití nástrojem kubectl je konfigurační soubor potřeba zkopírovat do lokace *~/.kube/config* a upravit jeho vlastnictví. To lze příkazy:

```
mkdir -p [userHome]/.kube
cp -i /etc/kubernetes/admin.conf [userHome]/.kube/config
chown [user]:[group] ~/.kube/config
```

Kde *user* odpovídá uživateli, který má být využit jako administrátor, *group* je jeho skupina a *userHome* je jeho domovský adresář. Po zkopírování konfiguračního souboru může daný uživatel používat kubectl ke spravování clusteru.

Po inicializaci je ještě potřeba nainstalovat zmíněný Network Addon, v případě této práce flannel. Díky tomu, že flannel běží v clusteru v podobě podu, je jeho instalace velmi jednoduchá. Stačí použít kubectl příkaz pro použití yaml manifestu z internetového odkaz. Jedná se o příkaz:

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

Tuto instalaci je potřeba udělat co nejdříve, dokud ještě nebyly přidané nějaké pody. Instalace po přidání podů může být mnohem komplikovanější.

5.2.2 Připojení nodu do clusteru

Pro připojení worker nodu ke clusteru pomocí kubeadm je potřeba ip adresa master nodu, discovery-token a discovery-token-ca-cert-hash. Obě lze získat přes master node.

Po inicializaci master nodu pomocí příkazu kubeadm init jsou tyto údaje automaticky vygenerovány a do terminálu je vytištěn formát kubeadm join příkazu s těmito údaji vloženými. Příkaz má formát:

```
kubeadm join --discovery-token <discovery-token> --discovery-token-ca-cert-hash sha256:<discovery-token-ca-cert-hash> <ip adresa master nodu>:6443
```

Po použití toho příkazu se spustí různé kontroly a pokud proběhnou v pořádku, tak se worker node připojí ke clusteru.

5.2.3 Připojení více master nodů

Pro existenci více master nodů v Kubernetes clusteru je nejprve potřeba load-balancer. To může být fyzický dedikovaný load-balancer, nebo load-balancer v podobě virtuálního stroje. Specifický postup nastavení load-balancer je pro každý specifické, ale ve všech případech je do něj nutné přidat ip adresy všech master nodů.

Připojení více master nodů probíhá podobně jako připojení worker nodů. Je ale nutné nejprve při inicializaci clusteru na prvním master nodu změnit konfigurační soubor. Je k němu potřeba přidat informaci, na které ip adrese se nachází load-balancer pro master nody. Upravený konfigurační soubor vypadá takto:

```
kind: ClusterConfiguration
apiVersion: kubeadm.k8s.io/v1beta3
networking:
 podSubnet: "10.244.0.0/16"
controPlaneEndpoint: "[lb-ip]:[lb-port]"
---
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
cgroupDriver: systemd
```

Kde *[lb-ip]* se ip adresa load-balanceru a *[lb-port]* je jeho port. Protože je k inicializaci clusteru použitý konfigurační soubor, nelze s *init* příkazem bohužel použít flag --upload-certs. Tato flag se postará o automatické nahrání certifikátů pro master nody. V případě nepoužití

této flag je potřeba certifikáty z prvního inicializovaného master nodu zkopírovat na ostatní master nody. Naštěstí lze příkaz *kubeadm init* aktivovat po jednotlivých fázích. Toho lze zde použít ke zpětnému nahrání certifikátů.

```
kubeadm init phase upload-certs –-upload certs
```

Po inicializaci clusteru s nastaveným controlPlaneEndpoint se vypíše *join* příkaz v podobném formátu jako ten pro worker nody, s tím rozdílem, že je potřeba přidat flag *--control-plane* a předat mu certifikátový klíč.

```
kubeadm join --discovery-token <discovery-token> --discovery-token-ca-cert-hash sha256:<discovery-token-ca-cert-hash> <adresa load-balanceru>:6443 --control-plane --certificate-key <certifikátový klíč>
```

V případě, že není příkaz s certifikátovým klíčem vypsán, nebo je potřeba nový klíč vygenerovat, lze použít příkaz:

```
kubeadm certs certificate-key
```

5.3 Povolení tvorby podů na master nodech

Plánování podů na master nody, jak bylo popsané v 3. kapitole, je omezené tzv. *taintem.* V tomto případě se jedná o taint s efektem *NoScheduling*. Tento taint lze ale odstranit. Lze toho docílit skrz příkaz přes *kubectl*. K odstranění i přidání taintu se využívá stejný příkaz: *kubectl taint*. Specifický formát příkazu je:

```
kubectl taint nodes <název nodu> <taint>
```

V případě odstranění tohoto taintu pro například node *master1* je příkaz v podobě:

```
kubectl taint nodes master1 node-role.kubernetes.io/control-plane:NoSchedule-
```

Po použití příkazu bude mít kube-scheduler možnost plánovat tvorbu aplikačních podů na master nodu *master1*.

5.4 Tvorba uživatelský účtů

Na rozdíl od účtů pro servisy (serviceaccount), účty pro uživatele neexistují jako objekt API a nelze je jednoduše skrz API vytvořit. Protože uživatelské účty fungují skrz autentizaci přes certifikáty, je potřeba daný certifikát prvně vygenerovat. To lze pomocí nástroje *OpenSSL*. V následovaných příkazech je potřeba zaměnit *<uživatel>* za jméno uživatele, který je vytvářen. Nejprve je potřeba vygenerovat soukromý klíč.

```
openssl genrsa -out <uživatel>.pem
```

Z tohoto klíče je dále potřeba vytvořit žádost pro podpis certifikátu.

```
openssl req -new -key <uživatel>.pem -out <uživatel>.csr -subj
"/CN=<uživatel>"
```

Poté je potřeba vytvořit *CertificateSigningRequest* objekt v Kubernetes clusteru. Toho lze docílit pomocít YAML souboru. Soubor žádosti podpisu certifikátu je potřeba do YAML souboru vložit zakódovaně skrz base64. Zakódovat žádost lze pomocí přikazu *base64*. Z výsledku je také potřeba oddělat nové řádky, aby se kód nacházel na jednom řádku. Toho lze docílit pomocí pipingu.

```
base64 <uživatel>.csr | tr -d '\n'
```

Dále je potřeba vytvořit zmíněný YAML soubor pro tvorbu CertificateSigningRequest objektu.

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
 name: user-request-<uživatel>
spec:
 groups:
 -system:authenticated
 request: <zakódovaná žádost>
 signerName: kubernetes.io/kube-apiserver-client
 expirationSeconds: 100000000
 usages:
 - digital signature
 - key encipherment
 - client auth
```

Pokud je tento soubor uložený pod jménem <uživatel>-csr.yaml, lze ho aplikovat v clusteru příkaze:

```
kubectl create -f <uživatel>-csr.yaml
```

Tuto žádost lze poté podepsat příkazem:

```
kubectl certificate approve user-request-<uživatel>
```

Nakonec je potřeba vytvořit z podepsaného certifikátu konfigurační soubor pro použití s kubectl. Nejprve se nastaví konfiguračnímu souboru informace o clusteru. Specificky jméno clusteru a adresa s portem API serveru. V defaultním nastavení se cluster jmenuje *kubernetes*.

```
kubectl –kubeconfig .kube/config-<uživatel> config set-cluster kubernetes –-server=https://<adresa API server>:<port API serveru>
```

Dále je konfiguračnímu souboru potřeba nastavit credentials, specificky certifikát a klíč. Příkaz také používá flag *–-embed-certs* pro embedování certifikátu a klíče do souboru v podobě textu pro snadnou přenositelnost. Bez této flag by konfigurační soubor odkazoval na jejich soubory.

```
kubectl –kubeconfig .kube/config-<uživatel> config set-credentials <uživatel> --client-certificate=<uživatel>-user.crt --client-key=<uživatel>.pem –-emded-certs=true
```

Nakonec je potřeba vytvořit pro konfigurační soubor context a nastavit ho jako momentálně využívaný. Příkaz pro tvorbu contextu:

```
kubectl --kubeconfig .kube/config-<uživatel> config set-context default --cluster=kubernetes –-user=<uživatel>
```

Příkaz pro nastavení contextu na momentálně používaný:

```
kubectl --kubeconfig .kube/config-<uživatel> config use-context default
```

Dále je potřeba nastavit nově vytvořenému uživateli pravomoci. K tomu slouží RBAC API objekty *Role, ClusterRole, RoleBinding a ClusterRoleBinding*. Role a RoleBinding se vztahují na specifický namespace, zatím co ClusterRole a ClusterRoleBinding se vztahují na celý cluster. Jako obvykle, tvorba těchto objektů probíhá přes YAML soubor. Protože v tomto příkladu je popsán YAML soubor pro tvorbu Role a RoleBinding, je potřeba vytvořit také namespace pro daného uživatele.

```
apiVersion: v1
kind: Namespace
metadata:
 name: <uživatel>-namespace
spec: {}
status: {}
```

Po vytvoření namespace pro uživatele je potřeba vytvořit jeho Role. V Role jsou určené jeho pravomoci. Pravomoci jsou určené v podobě vymezení API skupin, objektů a akcí. Pokud má mít uživatel možnost v daném Namespace spravovat všechny možnosti základních API objektů (například Pody a Deploymenty), tak se mu nastaví všechny akce u všech objektů ze skupin Core a Apps. Daný Role YAML soubor pak vypadá:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: <uživatel>
 namespace: <uživatel>-namespace
rule:
 - apiGroups: [“”,“apps“]
   resources: [“*“]
   verbs:[“*“]
```

Po vytvoření API objektu Role je ho ještě potřeba s uživatelem propojit. K tomu slouží objekt RoleBinding. Kvůli jeho použití pouze k propojení objektu Role s uživatelem je jeho YAML soubor konfigurace jednodušší oproti souboru Role. Stačí v něm určit Namespace a uživatele.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: <uživatel>
 namespace: <uživatel>-namespace
subjects:
 - kind: User
   name: <uživatel>
```

```
   apiGroups: rbac.authorization.k8s.io
roleRef:
   apiGroup: rbac.authorization.k8s.io
   kind: Role
   name: <uživatel>
```

Po vytvoření obou objektů lze konfigurační soubor využívat s nástrojem kubectl. Konfigurační soubor je díky embedovaným klíčům přenositelný. K jeho použití stačí počítač s nainstalovaným kubectl a připojením k API serveru. Přejmenováním a přesunutím do lokace *.kube/config* ho lze nastavit jako defaultní konfigurační soubor kubectl. V případě nezájmu nebo využití více konfiguračních souborů na jednom zařízení lze soubor použít skrz specifikaci jeho lokace příkaze:

```
kubectl --kubeconfig <lokace souboru>
```

6 AUTOMATIZOVANÉ SCRIPTY

Součástí této práce jsou scripty napsané autorem. Tyto scripty automaticky vykonají činnosti popsané v předešlé kapitole. Je zde popsaný vývoj a funkčnost scriptů pro tvorbu clusteru a uživatelských účtů.

6.1 Script pro tvorbu clusteru

6.1.1 Použití scriptu

Pro tento script je potřeba připravit počítače na stejné síti, které budou sloužit jako nody clusteru. Přesné požadavky jsou popsané v předešlé kapitole. Nejdůležitější požadavky jsou unikátní hostname, MAC adresa a product_uuid. Script byl testovaný na Linux distribuci Debian 12.5. Protože script využívá SSH pro práci s jednotlivými nody, je potřeba aby každý node měl nainstalovaný SSH server. U Debianu lze tohoto docílit při instalaci tím, že se v nabídce možných instalovatelných částí systému zvolí SSH server.

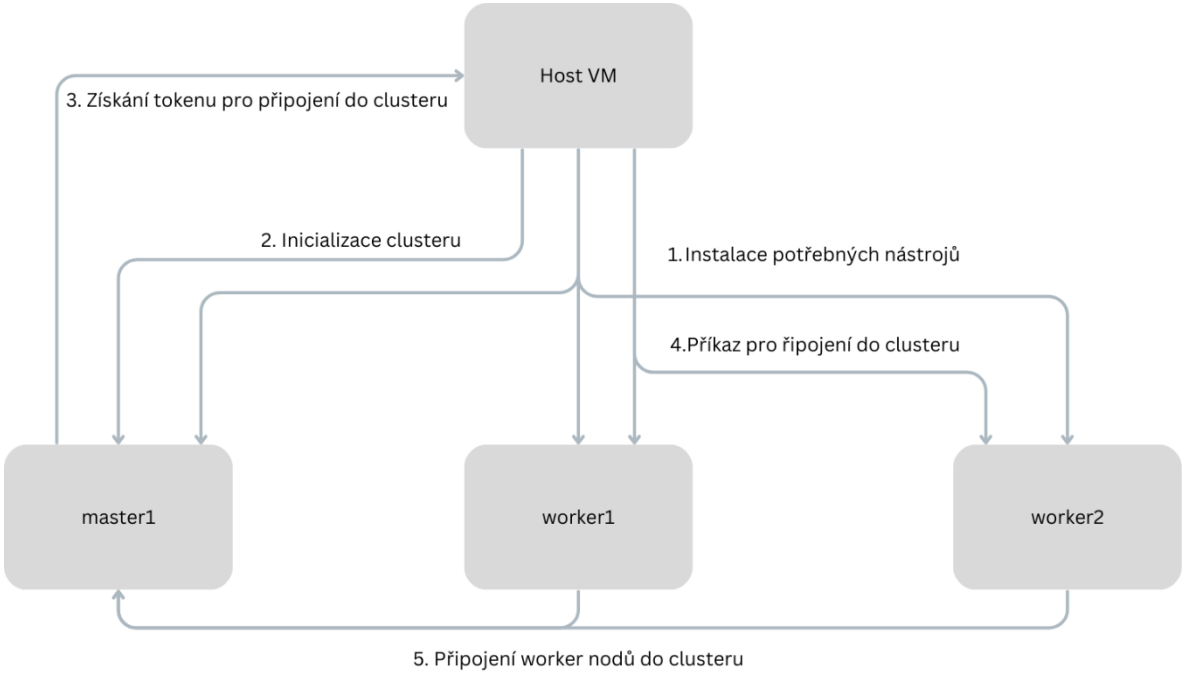
Další důležité požadavky pro počítač, na kterém je script spouštěný, je funkční klíčový pár, pomocí kterého se script může připojovat na nodové zařízení přes SSH. Účet, pro který je klíč potom nastavený, potřebuje sudo pravomoci a možnost používat sudo bez nutnosti hesla. Dát sudo pravomoci uživatelskému účtu tvořeného při instalaci operačního systému lze jednoduše nevytvořením root účtu při instalaci.

Protože se jedná o bash script, musí být spuštěný na počítači s operačním systémem Linux, který se nachází na stejné síti jako počítače určené pro nody. Scriptu musí být jako argument s flag **-f** předán textový soubor, který obsahuje informace o všech počítačích, kde každý jeden řádek souboru obsahuje informace o jednom z nich. Mezi tyto informace patří IP adresa a zdali se jedná o master nebo worker node. Tyto informace musí být odděleny mezerou. Přesný formát každého řádku je „**<ip-adresa> <worker-master>“.** Soubor musí být zakončený prázdným řádkem.

Kromě nutného argumentu v podobě souboru nodů, script podporuje možnosti tvorby cluster o více nodech a povolení tvorby podů na master nodech. Pokud je žádaná tvorba více master nodů, musí být script spuštěn s možností *-h loadbalancer*, kde *loadbalancer* odpovídá IP adrese loadbalanceru, který přesměrovává na dané master nody. Při tvorbě je také na počítači, na kterém je script spouštěn, nainstalovaný kubectl nástroj a administrační konfigurační soubor je na něj zkopírován. Pokud soubor nodů obsahuje více master nodů, ale není předána

tato možnost společně s IP adresou loadbalanceru, master nody budou mít nainstalované potřebné nástroje, ale bude vytvořen cluster pouze s prvním z nich.

Pokud je použítá možnost *-t*, bude při tvorbě odstraněn z master nodů taint zabraňující plánování podů na nich.



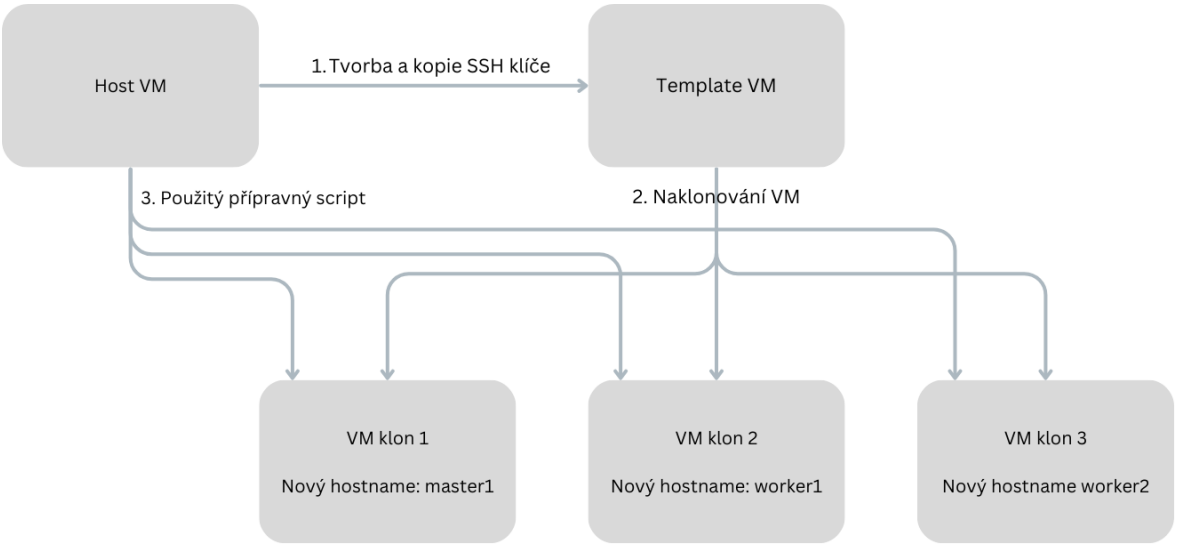
Obrázek 4. Průběh scriptu pro tvorbu clusteru

6.1.2 Script pro přípravu nodů

Kubernetes cluster se skládá z několika nodů. Všechny tyto nody mají podobné požadavky. V praxi se tedy dá očekávat, že místo jednotlivé instalace operačních systémů a jejich nastavení na každém nodu vytvoří administrátor jeden základní virtuální stroj, kde vše připravý, a poté tento virtuální stroj naklonuje na každý node. Tento přístup výrazně ulehčuje tvorbu clusteru, ale akce naklonování naklonuje i vlastnost nodů, která musí být unikátní, zejména hostname. Administrátor tedy stejně musí každému jednotlivému naklonovanému nodu věnovat pozornost a tuto vlastnost změnit.

Zde přichází na řadu script vytvořený autorem, jenž tyto akce automatizuje. Je očekávané, že se tento script použije před scripty ostatními. Přijímá jako argument stejný soubor, jako script pro tvorbu clusteru. Tento script využívá SSH stejným způsobem, jako ostatní scripty práce. Script se postupně připojí na každý z nodů a změní jejich hostname, podle toho, jestli se jedná o master nebo worker node.

Script funguje tak, že ve smyčce prochází všechny ip adresy počítačů předané skrz soubor v argumentu. U každého počítače navýší proměnou počtu (buďto *worker_count* nebo *master_count)*. Následně uloží původní hostname do proměnné a pomocí příkazu *hostnamectl set-hostname* změní hostname na nový, ve stylu *worker#* nebo *master#*, kde # odpovídá proměnné počtu. Script poté mění hostname v souboru */etc/hosts*. K tomu využivá předešlou proměnou s původním hostname. Pomocí příkazu *sed* nahradí původní hostname v souboru */etc/hosts* tím vytvořeným z proměnné počtu. Po této změně script počítač restartuje, aby změny proběhly.



Obrázek 5. Použití přípravného scriptu

6.1.3 Složení scriptu

Script se skládá z několika částí, které odpovídají krokům popsaných v předešlých kapitolách. Na začátku scriptu se kontroluje, jaké argumenty byly předané scriptu, buďto *-f, -h*, nebo *-t*. Pokud ano, tak se nastaví proměnné použité později ve scriptu. Pokud nebyl předaný argument *-f*, tedy soubor nodů, nebo se nejedná o soubor, tak se script ukončí.

Následují instalační funkce, které jsou poté volané později ve scriptu. Jedná se o funkce *master_install* a *worker_install*. Tyto funkce se starají o instalaci a nastavení všech požadavků pro master nody a worker nody respektive. Obě funkce jsou z velké části podobné, jejich rozdíly jsou popsané detailně v 5. kapitole. Funkce se starají o tvorbu iptables pravidle, stáhnutí a instalaci containerd, a instalaci kubectl, kubeadm a kubelet, podle typu nodu.

Následuje funkce pro inicializaci clusteru. Funkce obsahuje podmínky, které kontrolují, jestli je cluster tvořený s více master nody, nebo pouze jedním. Vytvoří kubeadm

konfigurační YAML soubor. Pokud se tvoří více master nodů, YAML soubor obsahuje ip adresu loadbalanceru, který slouží jako endpoint. Následuje samostatná inicializace clusteru pomocí konfiguračního souboru. Po inicializaci je příkaz pro připojení do clusteru uložený do proměnné. Pokud se tvoří cluster s více master nody, je zopakována fáze inicializace, ve které se nahrávají certifikáty pro cluster, jsou znovu vygenerované konfigurační soubory a je uložený příkaz pro připojení master nodu do clusteru. Poté je na hostitelském počítači nainstalován kubectl a z inicializovaného nodu zkopírován administrační konfigurační soubor. Pokud se tvoří více master nodů je v administračním konfiguračním souboru změněna adresa API serveru z IP adresy nodu na IP adresu loadbalanceru. Funkce je zakončená instalací flannelu.

Následují funkce pro připojení master nodu a funkce pro připojení worker nodu. Obě funkce jsou podobné. Funkce pro připojení worker nodu používá předtím uložený příkaz pro připojení do clusteru, zatím co funkce pro připojení master nodu používá příkaz pro připojení do clusteru jako master node.

Dále jsou ve scriptu smyčky, které tyto funkce volají. Jako první je smyčka pro instalaci potřebných nástrojů na nody. Smyčka prochází předaný soubor a podle toho, jestli je node master nebo worker, volá příslušnou funkci. Druhá smyčka je spuštěná po nainstalovaní všeho potřebného na nody. Ve smyčce se stejným způsobem jako u předešlý prochází jednotlivé nody. Pokud se jedná o první master node, volá se pro něj funkce pro inicializaci clusteru. U každého dalšího master nodu se kontroluje, jestli byla použita možnost *-h.* Pokud ano, je pro tyto nody volaná funkce pro připojení do clusteru jako master node. Třetí smyčka funguje stejně, s tím rozdílem že pro každý worker node volá funkce pro připojení do clusteru jako worker. Script je zakončen smyčkou která, pokud byla použita možnost *-t,* u každého z master nodů odstraní taint zabraňující plánování Podů na nich.

6.1.4 Vývoj scriptu

Na začátku vývoje bylo potřeba rozhodnout, jestli se bude jednat o jeden nebo pár velký script, nebo série více menších scriptů. Po konzultací s vedoucím práce bylo rozhodnuto pro tvorbu více větších scriptů. Vývoj ale stejně začal tvorbou více menších scriptů pro snazší testování a debuggování, s tím že se tyto menší scripty následně spojí do větších.

Jako první byl napsán script pro tvorbu iptables pravidel. Tento script byl následně původně testován na verzi Debian 11 Bullseye. V této verzi proběhla tvorba pravidel bez problému. Autor se ale později rozhodl pro použití novější verze distribuce Debian, zejména Debian 12

Bookworm. V této verzi již nástroj iptables není podporován vůbec. Script byl tedy přepsán na tvorbu pravidel skrz nástroj *nftables,* což je defaultní firewall této verze. Script byl dopsán do plné funkčnosti i s testováním. Poté bylo ale zjištěno, že Kubernetes ještě nepodporuje nftables, pouze iptables. Bylo tedy nutno vrátit se buďto ke starší verzi distribuce Debian, nebo iptables instalovat. Autor se rozhodl pro nadále použití verze 12 Bookworm a instalovat iptables na začátku scriptu.

Jako další script byl vyvinut ten pro aktivaci kernelových modulů a konfiguraci sysctl. V této části se píše do souborů ve vlastnictví uživatele *root* pod adresářem */etc*. Protože script pracuje skrz uživatele se sudo pravomocemi, ne root uživatele, bylo potřeba přijít na způsob, jako do souboru zapisovat. Klasický způsob zapsání do souboru, *cat [text] > soubor*, nepodporuje použití příkazu sudo. Zde autor přišel na způsob použití příkazu *tee* a pipingu. Nový příkaz byl napsán ve formátu *cat [text] | sudo tee soubor*. Tento způsob byl později ve scriptu použit pokaždé když bylo potřeba psát do root souborů.

Následoval script pro instalaci containerd. Zde bylo potřeba vytvořit script pro stáhnutí potřebných nástrojů z GitHubu a jejich instalaci. Bylo rozhodnuto pro vytvoření for smyčky, která prochází pole obsahující odkazy na tyto nástroje.

Poté byl vytvoření script pro instalaci kubectl, kubeadm a kubelet. Při vývoji tohoto scriptu byl jediný problém, jak zrušit SWAP. To lze sice příkazem *swapoff*, ale ten má vliv pouze do restartu počítače. Bylo potřeba upravit soubor */etc/fstab*. K tomu byl použit nástroj *sed*. Pomocí tohoto nástroje bylo v souboru */etc/fstab* přidán symbol # na začátek UUID disku.

Po vytvoření a ověření funkčnosti všech instalačních scriptů byly scripty skombinované do jednoho velkého, ze kterého se později stal script pro tvorbu clusteru. Bylo potřeba rozhodnout, jakým způsobem se budou předávat informace při spouštění scriptu. Z důvodu, že scriptu je potřeba přidat více informací o více nodech, bylo rozhodnuto o předání informací ve formě souboru oproti jednotlivých argumentech. Byla pro to přidaná while smyčka, která prochází jednotlivé řádky předaného souboru a pro každý řádek volá funkce.

Poté, co script obsahoval funkce pro instalaci nodů a smyčku která soubor prochází, bylo potřeba vytvořit funkci pro inicializaci clusteru. Začalo se s inicializací clusteru pro jeden master node. To proběhlo bez problémů. U inicializace cluster s více master nody byl problém. Protože je při inicializaci použit konfigurační soubor, nelze s příkazem *kubeadm init* použít možnost *--upload-certs.* Autor našel způsob, jak této možnosti dosáhnout i přes použití souboru, a to příkazem *kubeadm init phase upload-certs --upload-certs.* Tento příkaz

zopakuje fázi instalace, která by byla tou možností nastavena u původního příkazu. Bylo také potřeba znovu vygenerovat konfigurační souboru nodu, což se dosáhlo jejich smazáním a použití příkazu *kubeadm init phase kubeconfig all.*

Tvorba smyčky a funkce pro připojení nodů do clusteru proběhla bez problémů.

Poslední byla vytvořená smyčka, která se stará o oddělání taintu z master nodů. Zde bylo potřeba zjistit všechny hostname master nodů a pro každý volat příkaz. Autor zvažoval, že by ve smyčce prošel všechny ip adresy nodů a jejich hostname uložil do pole. Nakonec ale přišel se způsobem použití příkazu *kubectl get nodes* a kombinací nástrojů grep, sed a tr pro získání všech hostname master nodů v clusteru.

6.2 Script pro tvorbu uživatelských účtů

6.2.1 Použití scriptu

Tento script využívá kubectl a administrátorského konfiguračního souboru na hostitelském počítači, který byl zkopírován během tvorby clusteru. Podobně jako script pro tvorbu clusteru, využívá textového souboru pro předání informací scriptu. Př použití scriptu je potřeba předat lokaci souboru. Formát souboru musí být v podobě informací o daném uživateli na každém vlastní řádku a celý soubor nakonec zakončený prázdným řádkem. Každý řádek musí být napsán ve formátu **<jméno uživatele> <namespace> <API skupiny> <API objekty> <API akce>**. Jméno uživatele odpovídá jménu účtu, který se má vytvořit. Namespace odpovídá Namespace, ve kterém bude daný uživatel mít pravomoci. Namespace bude nově vytvořen. Pokud je jako Namespace určený *cluster*, tak bude mít uživatel pravomoci na celém clusteru. API skupiny, API objekty a API akce jsou podobné. U každého se specifikují skupiny, objekty a akce respektive, ke kterým má uživatel mít přístup. Může jich být určeno více, kde jsou jednotlivé rozdělené čárkou. Pokud je předané slovo *all*, určí script pravomoci uživateli všechny.

Příklad řádku pro uživatele *tomas* s namespace *tomas-ns* a všech pravomocí k API skupinám *core* a *apps*.

```
tomas tomas-ns core,apps all all
```

Příklad řádku pro uživatele *turecky* v namespace *turecky-ns* a pravomoci k vypsání Podů

```
turecky turecky-ns core pod get,list
```

Výsledkem scriptu je vytvořený kubectl konfigurační soubor, který dovoluje práci s clusterem jako daný uživatel.

6.2.2 Složení scriptu

Na začátku scriptu se nachází podmínka pro kontrolu předaných argumentů. Kontroluje, jestli byly argument 1 a jestli argument odpovídá souboru. Pokud podmínka selže, script se ukončí.

Následuje samotná funkce pro tvorbu uživatelů. Funkce generuje potřebný klíč a podpisovou žádost z něho. Tato žádost je vložena zakódovaná skrz Base64 do YAML souboru pro tvorbu *CertificateSigningRequest* objektu. Pomocí YAML souboru je objekt na clusteru vytvořen a hned akceptován. Z potvrzeného objektu je poté získán certifikát, který je dekódován a vložen do souboru. Z klíče a certifikátu je poté vytvořen kubectl konfigurační soubor pro uživatele. Dále jsou tvořeny potřebné RBAC objekty. Je zkontrolováno, jestli byl jako Namespace uživatele určený cluster nebo ne. Pokud ano, je vytvořen soubor pro tvorbu ClusterRole a ClusterRoleBinding, pokud ne, je vytvořen soubor pro tvorbu Namespace, Role a RoleBinding. Funkce je zakončená příkazem, který využívá nově vytvořeného souboru pro tvorbu daných objektů.

Poslední část scriptu obsahuje while smyčku, která prochází jednotlivé řádky předaného souboru a pro každý řádek volá předešlou funkci.

6.2.3 Vývoj scriptu

Na začátku vývoje bylo rozhodnuto, že seznam uživatelů bude předáván skrz soubor ze stejný důvodů, jako script pro tvorbu clusteru. Je totiž očekávané, že bude tvořeno více uživatelů zároveň.
Hlavním problém při tvorbě scriptu byl, jak získat pravomoci z předaného souboru. Jelikož je funkci předán řádek souboru oddělený po mezerách, bylo potřeba určit, jak předat více API skupin, objektů a akcí. Bylo rozhodnuto, že bude možné v souboru oddělit jednotlivé skupiny, objekty a akce čárkou. Ve funkci byla tedy vytvořeny while smyčky, které prochází jednotlivé části řádku, specificky smyčka pro skupiny, smyčka pro objekty a smyčka pro akce. Ve smyčkách bylo použito IFS pro určení čárky (,) jako dělení elementů pole. Smyčka prochází dané pole a jednotlivé elementy přidává do vlákna ve správném formátu pro YAML soubor. U každé smyčky bylo určené, že pokud pole obsahuje slovo *all*, smyčka skončí a jako jediný element ve vlákně zůstane hvězdička (*), protože reprezentuje

všechny možnosti. U smyčky pro skupiny se také kontroluje, jestli pole obsahuje skupinu *core*. Pokud ano, je do vlákna místo slova *core* vložen prázdný element, protože odpovídá core API skupině.

S vývojem zbytku scriptu nebyly žádné problémy.

7 OVĚŘENÍ FUNKČNOSTI SCRIPTŮ

V poslední kapitole této práce je potřeba ověřit správnou funkčnost, jak jednotlivých scriptů, tak clusteru, který tyto scripty vytvoří. Testování probíhalo na virtuálních strojích s operačním systémem GNU/Linux distribuce Debian verze 12.5 Bookworm. Tyto virtuální stroje byly spouštěné na operačním systému Windows 10 skrz software VirtualBox. Všechny virtuální stroje mají v nastavení sítě nastavený síťový adaptér na *Bridged Adapter*, aby mohly mezi sebou volně komunikovat. Všechny scripty jsou spouštěny na virtuálním stroji, který není součástí clusteru, je dále nazýván **hostitelský stroj**. Virtuální stroje, kromě toho, na kterém jsou scripty spouštěny, jsou vytvářeny klonováním virtuálního stroje, dále nazýván **template**. Při instalaci tohoto templatu byla zvolena instalace SSH serveru a *root* uživateli nebylo určené heslo. Díky tomu je první uživatel automaticky přidán do skupiny *sudo*. Skupině *sudo* pak bylo povoleno, aby používali příkaz *sudo* bez hesla. Tomuto uživateli byl poté z hlavního počítače zkopírován veřejný klíč pro SSH, aby se šlo do toho uživatele přes SSH přihlásit z hlavního počítače bez hesla.

Testování je rozděleno na 5 části, kde každá odpovídá jednomu použití scriptů. Testování je doplněné obrázky, které ukazují stav během testování.

7.1.1 Použití přípravného scriptu na naklonovaných virtuálních strojích

V této části se testuje funkcionalitu přípravného scriptu pro změnu hostname virtuálních počítačů. Očekávaný výsledek byl, že budou po použití scriptu všechny počítače přejmenované podle toho, jestli se jedná o master nebo worker node, a tento hostname změněn v souboru */etc/hosts*.

Pro testování byly naklonované 4 virtuální stroje z templatu. Dva z nich pro určení jako master node a dva jako worker node. Protože byly naklonované z templatu, který má defaultní hostname *debian*, všechny virtuální stroje tento hostname sdílí.

U všech virtuálních strojů bylo nejprve zjištěné, jaké IP adresy jim byl přidělené. Na to byl použit příkaz *ip a*.

```
zutyro@debian:~$ ip a
inet 192.168.0.124/24 brd 192.168.0.255 scope
zutyro@debian:~$ ip a
 inet 192.168.0.192/24 brd 192.168.0.255 scope
zutyro@debian:~$ ip a
inet 192.168.0.81/24 brd 192.168.0.255 scope
zutyro@debian:~$ ip a
 inet 192.168.0.80/24 brd 192.168.0.255 scope
```

Obrázek 6. IP adresy nodů

Tyto ip adresy byly vložené do souboru *ipadresy.txt* společně s upřesněním, jestli se jedná o master nebo worker node. Poslední řádek souboru byl ponechán prázdný, aby byl soubor správně přečten.

```
Scripts > ipadresy.txt
1   192.168.0.124 master
2   192.168.0.192 master
3   192.168.0.81 worker
4   192.168.0.80 worker
5
```

Obrázek 7. Soubor ipadresy.txt

Na hostitelském stroji byl poté testovaný script *priprava_nodu.sh* spuštěný a byl mu předám tento soubor. Byl spuštěn ve formátu *bash priprava_nodu.sh ipadrexy.txt*. Script se úspěšně připojil na IP adresy napsané v souboru. Po restartování virtuálních strojů scriptem měli nody úspěšně změněné jak hostname systému, tak hostname v soubor */etc/hosts*.

```
zutyro@master1:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       master1
zutyro@master2:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       master2
zutyro@worker1:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       worker1
zutyro@worker2:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       worker2
```

Obrázek 8. Hostname a soubory */etc/hosts*

Testování proběhlo úspěšně a podle očekávání.

7.1.2 Tvorba clusteru s jedním master nodem

Tato část testovala funkcionalitu hlavního scriptu pro tvorbu clusteru. K testování byly využity 3 virtuální stroje na kterých byl spuštěn přípravný script. Testovala se tvorba clusteru

s 1 master nodem a 2 worker nody. Bylo očekávané, že script vytvoří z těchto nodů cluster , kde bude možné úspěšně vytvořit Deployment a Service co na jeho Pody odkazuje.

Byly zjištěny jejich ip adresy a ty sepsané do souboru *ipadresy.txt* jako v předešlém testování. Byl použit přípravný script pro změnu jejich hostname po naklonování. Po jejich restartováním scriptem byl soubor *ipadresy.txt* předán scriptu *cluster_install.sh.*

Forma příkazu pro použití scriptu byla *bash cluster_install.sh -f ipadresy.txt.* Po ukončení běhu scriptu bylo pomocí nově nainstalovaného nástroje *kubectl* na hostitelském virtuálním stroji a zkopírovaným administračním konfiguračním souborem z master nodu zjištěno, že byly úspěšně všechny 3 nody přidané do clusteru. Byl využit příkaz *kubectl get nodes*.

```
zutyro@debian:~$ kubectl get nodes
NAME      STATUS   ROLES           AGE   VERSION
master1   Ready    control-plane   69s   v1.29.4
worker1   Ready    <none>          30s   v1.29.4
worker2   Ready    <none>          21s   v1.29.4
```

Obrázek 9. Nody ve vytvořeném clusteru

Podle obrázku lze vidět, že byl nody správně přidané do clusteru a jsou ve stavu *Ready*. Dále bylo ověřeno, že všechny systémové pody jsou spuštěné v pořádku, včetně podů flannelu. Byl na to využit příkaz *kubectl get pods --all-namespaces*.

```
zutyro@debian:~$ kubectl get pods --all-namespaces
NAMESPACE      NAME                                READY   STATUS    RESTARTS   AGE
kube-flannel   kube-flannel-ds-998km               1/1     Running   0          17m
kube-flannel   kube-flannel-ds-vqbnc               1/1     Running   0          17m
kube-flannel   kube-flannel-ds-zftz8               1/1     Running   0          17m
kube-system    coredns-76f75df574-cvfpn            1/1     Running   0          17m
kube-system    coredns-76f75df574-ss5qc            1/1     Running   0          17m
kube-system    etcd-master1                        1/1     Running   1          18m
kube-system    kube-apiserver-master1              1/1     Running   1          17m
kube-system    kube-controller-manager-master1     1/1     Running   1          17m
kube-system    kube-proxy-jfcsg                    1/1     Running   0          17m
kube-system    kube-proxy-s2rvh                    1/1     Running   0          17m
kube-system    kube-proxy-zphp9                    1/1     Running   0          17m
kube-system    kube-scheduler-master1              1/1     Running   1          17m
```

Obrázek 10. Systémové pody běží v pořádku

Na obrázku jde vidět, že všechny systémové pody běží v pořádku.

Následovalo testování tvorby a funkčnosti Deploymentu a Servicu pro něj. K testování byla použita jednoduchá aplikace v Node.js, která vytvoří server na portu 8080 a odpovídá na požadavky jednoduchou správou s ID kontejneru. Aplikace byla převzatá z jednoho zdroje práce. [4]

```
Scripts > testapp > JS testapp.js > ...
1   const http = require('http');
2   const os = require('os');
3
4   console.log('Server startuje')
5
6   var server = function (zadost, odpoved) {
7       console.log('Obdržen request z ' + zadost.connection.remoteAddress);
8       odpoved.writeHead(200);
9       odpoved.end('Dosahnut server ' + os.hostname() + '\n');
10  };
11
12  var www = http.createServer(server);
13  www.listen(8080);
```

Obrázek 11. Aplikace pro testování Deploymentu

Z této aplikace byl poté vytvořen container obraz pomocí nástroje Docker a nahrán na Docker Hub. Vytvořený byl souborem *Dockerfile*, který aplikaci *testapp.js* přidal do obrazu *node:7* a spustil ji v něm. Soubor by vytvořen se stejném adresáři jako *testapp.js*.

```
Scripts > testapp > Dockerfile
1   FROM node:7
2   ADD testapp.js /testapp.js
3   ENTRYPOINT ["node", "testapp.js"]
```

Obrázek 12. Dockerfile testové aplikace

Obraz byl vytvořen příkazem *sudo docker build -t testapp .* spuštěným ve stejném adresáři jako *testapp.js* a *Dockerfile*. Po vytvoření obrazu mu bylo potřeba pro nahrání na Docker Hub přidat tag uživatele, specificky *zutyro.* Toho se dosáhl přikazem *sudo docker tag testapp zutyro/testapp*. Po přihlášení na autorův účet příkazem *sudo docker login* byl obraz nahrán na Docker Hub příkazem *sudo docker push zutyro/testapp*.

Po vytvoření testovací aplikace a jejího nahrání na Docker Hub bylo možné přejít na tvorbu samotného Deploymentu a Servicu. Pro jejich tvorbu bylo použitu YAML souborů. V tomto soubor byl definován jak Deployment, tak Service. V definici Deploymentu byl určený jako obraz containeru *zutyro/testapp* a nastavena tvorba 4 replicas. Service byl určený jako NodePort, aby šlo Pody lehce otestovat.

Scripts > ! testdeployment.yaml > {} spec > [ ] ports > {} 0 > # nodePort

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-deployment
  labels:
    app: testapp
spec:
  replicas: 4
  selector:
    matchLabels:
      app: testapp
  template:
    metadata:
      labels:
        app: testapp
    spec:
      containers:
      - name: test-container
        image: zutyro/testapp
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: test-service
spec:
  type: NodePort
  selector:
    app: testapp
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 30000
```

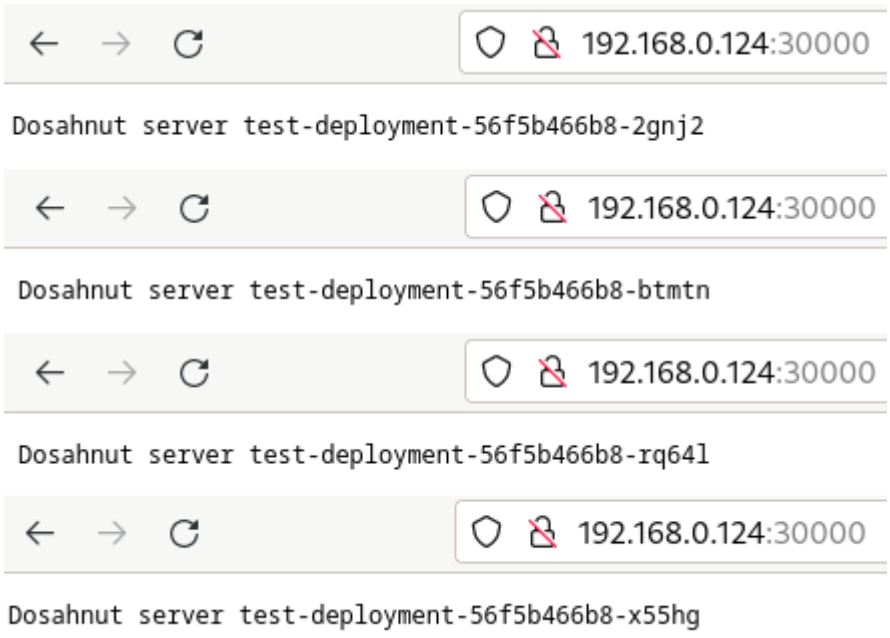
Obrázek 13. YAML soubor Deploymentu a Servicu

Deployment a Service byly na clusteru vytvořeny příkazem *kubectl apply -f testdeployment.yaml*. Deployment a Service byly úspěšně vytvořeny a Pody naplánované na worker nody, kde byly úspěšně spuštěné.

```
zutyro@debian:~/Scripts$ kubectl get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
test-deployment   4/4     4            4           9m34s
zutyro@debian:~/Scripts$ kubectl get services
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP          13h
test-service   NodePort    10.100.12.224   <none>        8080:30000/TCP   9m38s
zutyro@debian:~/Scripts$ kubectl get pods -o=custom-columns=NAME:.metadata.name
NAME                               STATUS    NODE
test-deployment-56f5b466b8-2gnj2   Running   worker2
test-deployment-56f5b466b8-btmtn   Running   worker1
test-deployment-56f5b466b8-rq64l   Running   worker1
test-deployment-56f5b466b8-x55hg   Running   worker2
```

Obrázek 14. Stav Deploymentu, Servicu a Podů

Protože byl jako typ Servicu určený NodePort s portem 30000, funkčnost Podů byla testováná připojením na master node na tomto portu skrz webový prohlížeč. Skrz prohlížeč se podařilo úspěšně připojit na každý z testových Podů.



Obrázek 15. Úspěšné připojení na Pody

Testování proběhlo úspěšně a podle očekávání. Script vytvořit Kubernetes Cluster s 1 master nodem a 2 worker nody. Bez problémů šel vytvořit testový Deployment a podařilo se připojit na ním vytvořené Pody skrz Service.

7.1.3 Tvorba clusteru s třemi master nody

V této části se znovu testovala funkcionalitu hlavního scriptu pro tvoření clusteru. Na rozdíl od předešlého testování se zde testovala tvorba cluster s více master nody. Přesněji 3 master

nody a 2 worker nody. Očekávaný výsledek byl, že script v pořádku vytvoří cluster se 3 master nody a 2 worker nody. S clusterem bude možné pracovat přes kubectl skrz loadbalancer, bez problému půjde vytvořit testový Deployment a Service, a jejich Pody budou fungovat.

Byl znovu použit přípravný script. Jako loadbalancer slouží virtuální stroj s nainstalovaným nástrojem HAProxy. V konfiguračním souboru *haproxy.cfg* bylo nastavené přesměrování na ip adresy master nodů ve stylu round robin.

```
frontend kubernetes
        bind 192.168.0.236:6443
        option tcplog
        mode tcp
        default_backend kubernetes-master-nodes

backend kubernetes-master-nodes
        mode tcp
        balance roundrobin
        option tcp-check
        server kubernetes-master1 192.168.0.125:6443 check fall 3 rise 2
        server kubernetes-master2 192.168.0.248:6443 check fall 3 rise 2
        server kubernetes-master3 192.168.0.143:6443 check fall 3 rise 2
```

Obrázek 16. Přesměrování na master nody v souboru *haproxy.cfg*

Na hostitelském stroji byl script spuštěný s možností *-h* s předaným souborem *ipadresy.txt* jako v předešlém testu. Příkaz byl ve formě *bash cluster_install.sh -h 192.168.0.236 -f ipadresy.txt*. Po ukončení běhu scriptu bylo příkazem *kubectl get nodes* ověřeno, že byl cluster v pořádku vytvořen. Všechny nody byly ve stavu *Ready*.

```
zutyro@debian:~/Scripts$ kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
master1   Ready    control-plane   7m40s   v1.29.4
master2   Ready    control-plane   5m35s   v1.29.4
master3   Ready    control-plane   3m4s    v1.29.4
worker1   Ready    <none>          2m47s   v1.29.4
worker2   Ready    <none>          2m35s   v1.29.4
```

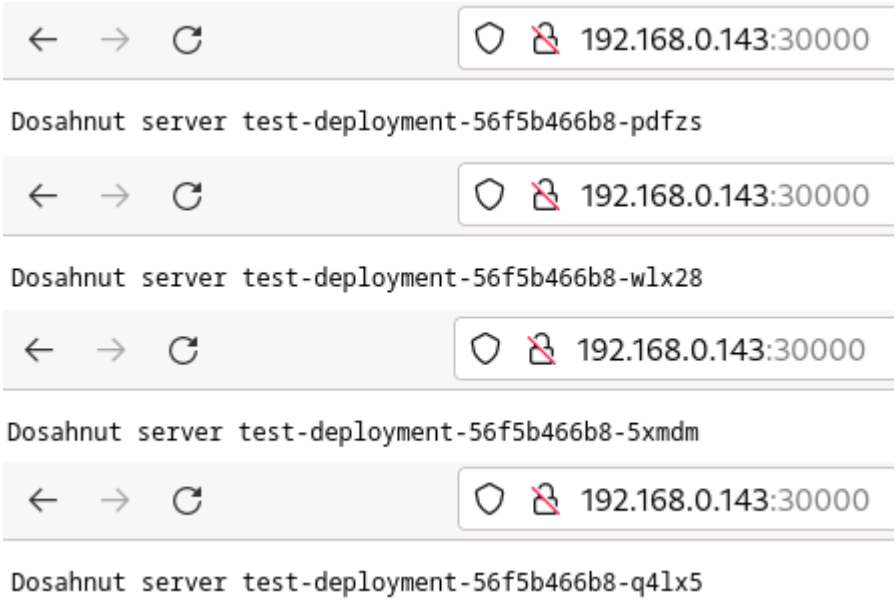
Obrázek 17. Nody v clusteru s více master nody

Po ověření stavu clusteru se přešlo na testování tvorby Deploymentu a Servicu. Pro testování byl použit stejný YAML soubor a aplikace jako v předešlém testování. Deployment, Service a Pody byly vytvořené v pořádku.

```
zutyro@debian:~/Scripts$ kubectl get pods -o=custom-columns=NAME:.metadata.nar
NAME                               STATUS    NODE
test-deployment-56f5b466b8-5xmdm   Running   worker2
test-deployment-56f5b466b8-pdfzs   Running   worker1
test-deployment-56f5b466b8-q4lx5   Running   worker2
test-deployment-56f5b466b8-wlx28   Running   worker1
zutyro@debian:~/Scripts$ kubectl get deployments
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
test-deployment   4/4     4            4           15m
zutyro@debian:~/Scripts$ kubectl get services
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP          52m
test-service   NodePort    10.98.100.238   <none>        8080:30000/TCP   15m
```

Obrázek 18. Stav Deploymentu, Serviceu a Podů v clusteru třetího testování

Funkčnost Podů byla opět ověřená připojením skrz webový prohlížeč. Podařilo se připojit na všechny Pody bez problému.



Obrázek 19. Úspěšné připojení na Pody v clusteru s více master nody

Testování proběhlo podle očekávání. Script úspěšně vytvořil Kubernetes cluster s více master nody. Použití kubectl přes loadbalancer fungovalo bez problému, stejně jako tvorba Deploymentu a Servicu.

7.1.4 Tvorba clusteru s možností tvorby podů na master nodech

Tato část je podobná předešlým testováním funkčnosti clusteru. S tím rozdílem, že se při tvorbě clusteru povolila možnost tvorby podů na master nodech. Byla testována tvorba clusteru se 3 master nody bez worker nodů. Očekávaný výsledek byl ten, že po přidání master nodů do clusteru jim bude oddělán taint, který zabraňuje tvorbě Podů na nich, a Pody se na

nich v pořádku vytvoří. Tvorba Deploymentu, Servicu a funkcionalita Podů už byla plně testovaná u předešlých testů.

Jako u předešlého testování byl použit přípravný script a loadbalancer HAProxy. Instalační script byl použit s možností *-t* ve formě *bash cluster_install.sh -h 192.168.0.236 -t -f ipadresy.txt*. Cluster byl vytvořený úspěšně, což odpovídá předešlému testování. Po vytvoření clusteru bylo zkontrolované, že master nodům byl taint odstraněný. Na to byl použit příkaz *kubectl describe node*. Bylo zjištěno, že se všech master nodů byl taint v pořádku odstraněn.

```
zutyro@debian:~/Scripts$ kubectl describe nodes
Name:               master1
Roles:              control-plane
CreationTimestamp:  Thu, 25 Apr 2024 18:49:51 +0200
Taints:             <none>
Unschedulable:      false
Name:               master2
Roles:              control-plane
CreationTimestamp:  Thu, 25 Apr 2024 18:52:33 +0200
Taints:             <none>
Unschedulable:      false
Name:               master3
Roles:              control-plane
CreationTimestamp:  Thu, 25 Apr 2024 18:54:36 +0200
Taints:             <none>
Unschedulable:      false
```

Obrázek 20. Master nody s odstraněným taintem

Poté bylo ověřeno, že se Pody skutečně na master nody naplánují. Byl na to použitý předešlý soubor pro tvorbu Deploymentu a Servicu příkazem *kubectl apply -f testdeployment.yaml.* Deployment a Service byl vytvořen v pořádku, což souhlasí s předešlým testováním. Upraveným příkazem *kubectl get pods* bylo poté zjištěno, že Pody byly skutečně vytvořeny na master nodech.

```
zutyro@debian:~/Scripts$ kubectl get pods -o=custom-cc
NAME                                STATUS    NODE
test-deployment-56f5b466b8-bpdmk    Running   master1
test-deployment-56f5b466b8-g7xf6    Running   master2
test-deployment-56f5b466b8-mc7dh    Running   master3
test-deployment-56f5b466b8-xpsrm    Running   master3
```

Obrázek 21. Pody vytvořené na master nodech

Testování bylo úspěšné a podle očekávání. Poté co script vytvořil cluster byl master nodům oddělaný taint bránící tvorbě Podů na nich. Po tvorbě testového Deploymentu byly Pody skutečně vytvořené na master nodech.

7.1.5 Tvorba uživatelský účtů

V posledním testování se kontrolovala funkčnost scriptu pro tvorbu nových uživatelských účtů na clusteru. Byla testována tvorba 4 různých uživatelských účtů. První uživatel by měl mít všechny pravomoce na celém clusteru, druhý uživatel pravomoce na vypání Podů v defaultním Namespace, třetí uživatel tvorbu a vypsání Deploymentů ve vlastním Namespace a čtvrtý uživatel tvorbu, vypsání a mazání Serviců a Podů ve vlastním Namespace. Očekávaný výsledek testování je, že script vytvoří všechny zadané uživatelské účty se správnými pravomocemi.

Byl vytvořen cluster s 1 master nodem a 2 worker nody. Poté byl vytvořen soubor obsahující požadované uživatelské účty se jménem *uzivatele.txt*.

Scripts > uzivatele.txt

```
1  user1 cluster all all all
2  user2 default core pods get,list
3  user3 user3-ns apps deployments get,list,create
4  user4 user4-ns core pods,services get,list,create,delete
5
```

Obrázek 22. Soubor *uzivatele.txt*

S tímto souborem byl spuštěn testovaný script. Byl spuštěn příkazem formou *bash add_user.sh uzivatele.txt.* Script vytvořil potřebné certifikáty a konfigurační soubory pro uživatele.

```
zutyro@debian:~$ ls | grep -i 'user'
user1-clusterrbac.yaml
user1.csr
user1-csr.yaml
user1.pem
user1-user.crt
user2.csr
user2-csr.yaml
user2.pem
user2-rbac.yaml
user2-user.crt
user3.csr
user3-csr.yaml
user3.pem
user3-rbac.yaml
user3-user.crt
user4.csr
user4-csr.yaml
user4.pem
user4-rbac.yaml
user4-user.crt
zutyro@debian:~$ ls .kube/ | grep -i "user"
config-user1
config-user2
config-user3
config-user4
```

Obrázek 23. Certifikáty a konfigurační soubory uživatelů

S konfiguračními soubory vytvořenými se mohlo přejit na testování pravomocí těchto uživatelů. První uživatel byl vytvořen se všemi pravomocemi v clusteru. Tyto pravomoce se testovali vypsáním nodů clusteru příkazem *kubectl --kubeconfig .kube/config-user1 get nodes*. To se podařilo bez problému.

```
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user1 get nodes
NAME      STATUS   ROLES           AGE   VERSION
master1   Ready    control-plane   59m   v1.29.4
worker1   Ready    <none>          55m   v1.29.4
worker2   Ready    <none>          55m   v1.29.4
```

Obrázek 24. Testování pravomocí uživatele 1

Druhý uživatel byl vytvořen s pravomocí výpisu Podů v Namespace default. To se otestovalo výpisem Podů a výpisem Deploymentů. Uživatel správně mohl vypsat Pody a nemohl vypsat Deploymenty.

```
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user2 get pods
No resources found in default namespace.
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user2 get deployments
Error from server (Forbidden): deployments.apps is forbidden: User "user
```

Obrázek 25. Testování pravomocí uživatele 2

Třetí uživatel byl vytvořen s pravomocí vytvářet a vypisovat Deploymenty ve vlastním Namespace, v tomto případě user3-ns. K tomu byl použit předtím vytvořený soubor pro tvorbu testového Deploymentu. Deployment byl vytvořen v Namespace user3-ns úspěšně a podařilo se ho vypsat. Pokus o výpis Deploymentů v Namespace default naopak neuspěl. Pravomoci měl uživatel tedy nastavené správně.

```
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user3 apply -f Scripts/testdeploy
deployment.apps/test-deployment created
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user3 get deployments -n user3-ns
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
test-deployment   0/4     4            0           12s
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user3 get deployments
Error from server (Forbidden): deployments.apps is forbidden: User "user3" cannot li
```

Obrázek 26. Testování pravomocí uživatele 3

Poslední uživatel byl vytvořen s pravomocí tvořit, vypisovat a odstraňovat Pody a Servicy ve vlastním Namespace, tedy user4-ns. Pro testování byl vytvořen jednoduchý Pod a Service pouze pro testování pravomocí. Testovala se jejich tvorba, výpis a odstranění. To proběhlo v pořádku.

```
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user4 apply -f Scripts/testpod
pod/test-pod created
service/test-service created
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user4 get pods -n user4-ns
NAME       READY   STATUS    RESTARTS   AGE
test-pod   1/1     Running   0          53s
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user4 get services -n user4-ns
NAME           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
test-service   NodePort   10.107.65.182   <none>        8080:30000/TCP   59s
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user4 delete services test-ser
service "test-service" deleted
zutyro@debian:~$ kubectl --kubeconfig .kube/config-user4 delete pods test-pod -n
pod "test-pod" deleted
```

Obrázek 27. Testování pravomocí uživatele 4

Testování scriptu pro tvorby uživatelů proběhlo v pořádku a bez problémů. Script správně vytvořil všechny požadované uživatele s jejich pravomocemi. Vytvořené konfigurační soubory šli bez problému používat s nástrojem kubectl k práci s clusterem.

ZÁVĚR

V této práci jsou probrané existující práce v dané oblasti a jejich rozdíly oproti práce této. Probrané jsou také použité technologie v práci společně s koncepty virtualizace a kontejnerizace a jejich rozdíly. Jsou probrané možnosti tvorby clusteru, ze kterých byl určen způsob, který se automatizoval. Jsou také určené úpravy clusteru, které se automatizovali jako součást tvorby clusteru. Dále jsou popsané možné komponenty, ze kterých autor po zkoušení určil specifické, které byly v práci nakonec použity.

V práci se podařilo jak určení jednotlivých kroků potřebných k tvorbě Kubernetes clusteru, tak jejich automatizace. Podle kroků pro tvorbu a úpravu Kubernetes clusteru určených v začátku praktické částí práce byly úspěšně vytvořeny 3 scripty pro shell bash. Jedná se o script pro přípravu nodů, script pro tvorbu clusteru a script pro tvorbu uživatelských účtů. Scripty byly testovány a úspěšně splňují vše, co se od nich očekávalo.

Použitím scriptů lze úspěšně vytvořit plnohodnotný Kubernetes cluster na lokálních zařízení s více uživatelskými účty. Script pro tvorbu clusteru dokáže vytvořit jak cluster s jedním master nodem, tak cluster s více master nody a dokáže povolit tvorbu Podů na master nodech. Script pro tvorbu uživatelských účtů zvládá vytvořit více uživatelských účtu, jejich pravomoci clusteru a jejich konfigurační soubory pro kubectl.

Autora napadly dvě možné cesty, jak práci v budoucnu rozšířit. Jedna z nich, je rozšíření samotné tvorby a nastavení Kubernetes clusteru. V této práci byly vybrané specifické komonenty, pomocí kterých je cluster automaticky vytvořen. Script pro tvorbu clusteru by mohl být rozšířený o možnosti výběru komponentů uživatelem, kde by se script volbám přizpůsobil a vytvořil cluster podle nich. Script by také mohl nabídnout rozšířené možnosti co se týče konfigurace inicializace clusteru.

Druhá cesta, je rozšíření práce o více scriptů zabývajících se automatizací akcí na již vytvořeném clusteru. Tvorba uživatelských účtů, kterou se zabývá jeden ze scriptů práce, lze považovat za jednu z nich. Práce by mohla být rozšířená o scripty pro automatickou tvorbu různých API objektů podle zadání uživatele na podobný způsob, jako již zmíněny script pro tvorbu uživatelských účtů. Příkladem by mohl být script, který automaticky vytvoří Deployment a Service pro jeho Pody zároveň. Script pro tvorbu uživatelských účtů by mohl být také rozšířený. Script momentálně vytváří nové uživatelské účty podle zadání uživatele. Mohl by být rozšířený o možnost přidání nových pravomocí již existujícím uživatelů, nebo možnost jejich účty z clusteru odstranit.

SEZNAM POUŽITÉ LITERATURY

[1] D. Bringhenti, R. Sisto and F. Valenza, "Security automation for multi-cluster orchestration in Kubernetes," *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, Madrid, Spain, 2023, pp. 480-485, doi: 10.1109/NetSoft57336.2023.10175419.

[2] *Kubernetes with Kubeadm: Fully Automated Installation with Terraform*. Online. DEV Community. DEV Community © 2016 - 2024. Dostupné z: . [cit. 2024-04-05].

[3] N. Astyrakakis, Y. Nikoloudakis, I. Kefaloukos, C. Skianis, E. Pallis and E. K. Markakis, "Cloud-Native Application Validation & Stress Testing through a Framework for Auto-Cluster Deployment," *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Limassol, Cyprus, 2019, pp. 1-5, doi: 10.1109/CAMAD.2019.8858164.

[4] LUKŠA, Marko. Kubernetes in Action. New York: Manning, 2017. ISBN 9781617293726.

[5] WALKER, James. What Is containerd, And How Does It Relate to Docker and Kubernetes? Online. 2021, s. 1. Dostupné z: How-To Geek, https://www.howtogeek.com/devops/what-is-containerd-and-how-does-it-relate-to-docker-and-kubernetes/. [cit. 2024-03-01].

[6] DOCKER INC. *Overview of the get started guide*. Online. DOCKER INC. Docker Docs. C2013-2024. Dostupné z: https://docs.docker.com/get-started/. [cit. 2024-03-01].

[7] BAIER, Jonathan a Jesse WHITE. Getting Started with Kubernetes: Extend your containerization strategy by orchestrating and managing large-scale container deployments. 3rd edition. Birmingham, UK: Packt Publishing, 2018. ISBN 978-1-78899-472-9.

[8] JAIN, Piyush. Kubernetes Installation Options: The Hard Way, Kubedm, MiniKube, Managed K8s (EKS, AKS, OKE, GKE). Online. *K21Academy*. 2023, s. 1. Dostupné z: https://k21academy.com/docker-kubernetes/kubernetes-installation-options/. [cit. 2024-03-01].

[9] DOCKER INC. *Deploy on Kubernetes with Docker Desktop*. Online. DOCKER INC. Docker Docs. C2013-2024. Dostupné z: //docs.docker.com/desktop/kubernetes/. [cit. 2024-03-01].

[10] *What Is Minikube?* Online. SYSDIG, INC. Sysdig. C2024. Dostupné z: https://sysdig.com/learn-cloud-native/kubernetes-101/what-is-minikube/. [cit. 2024-03-01].

[11] THE KUBERNETES AUTHORS. *Kubeadm*. Online. THE KUBERNETES AUTHORS. Kubernetes. C2024. Dostupné z: https://kubernetes.io/docs/reference/setup-tools/kubeadm/. [cit. 2024-03-01].

[12] *KOps - Kubernetes Operations*. Online. KOps - Kubernetes Operations. C2024. Dostupné z: https://kops.sigs.k8s.io. [cit. 2024-03-01].

[13] LUCAS, Michael W. *SSH Mastery: OpenSSH, PuTTY, Tunnels, and Keys*. 2nd ed. Tilted Windmill Press, 2018. ISBN 9781642350029.

[14] RAMEY, Chet a FOX, Brian. *Bash Reference Manual*. Online. Edition 5.2. Free Software Foundation, 2022. Dostupné z: https://www.gnu.org/software/bash/manual/bash.pdf. [cit. 2024-03-15].

[15] HERTZOG, Raphaël a MAS, Roland. *The Debian Administrator's Handbook*. Lulu Press, 2020. ISBN 9791091414197.

[16] *Iptables Tutorial: Ultimate Guide to Linux Firewall*. Online. PhoenixNAP. 2020. Dostupné z: https://phoenixnap.com/kb/iptables-tutorial-linux-firewall. [cit. 2024-04-08].

[17] *Comparing Kubernetes CNI Providers: Flannel, Calico, Canal, and Weave*. Online. Rancher. © 2023. Dostupné z: https://www.suse.com/c/rancher_blog/comparing-kubernetes-cni-providers-flannel-calico-canal-and-weave/. [cit. 2024-04-15].

[18] *Docker vs containerd vs CRI-O: An In-Depth Comparison*. Online. PhoenixNAP. ©2024. Dostupné z: https://phoenixnap.com/kb/docker-vs-containerd-vs-cri-o. [cit. 2024-04-07].

[19] *Mirantis / cri-dockerd: dockerd as a compliant Container Runtime Interface for Kubernetes*. Online. GitHub. © 2024. Dostupné z: https://github.com/Mirantis/cri-dockerd. [cit. 2024-04-07].

[20] *Mirantis Product Overview*. Online. Mirantis Documentation Portal. © 2005 - 2023. Dostupné z: https://docs.mirantis.com/mcr/20.10/overview.html. [cit. 2024-04-08].

[21] *Virtualization vs. Containerization — Comparing Differences*. Online. Liquid Web. 2023. Dostupné z: https://www.liquidweb.com/kb/virtualization-vs-containerization/. [cit. 2024-05-07].

[22] *A Crash Course in Kubernetes*. Online. ByteByteGo Newsletter. 2023. Dostupné z: https://blog.bytebytego.com/p/a-crash-course-in-kubernetes. [cit. 2024-05-07].

[23] *What is Kubernetes? Uses, Features, Architecture & Working*. Online. Cloudwithease. © 2024. Dostupné z: https://cloudwithease.com/what-is-kubernetes/. [cit. 2024-05-07].

[24] RANI, Osnat. *10 Kubernetes Alternatives and Why You Need Them*. Online. Cloud Native Wiki. © 2024. Dostupné z: https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-alternatives/. [cit. 2024-05-07].

[25] WALKER, James. *What is Docker Swarm Mode and When Should You Use It?* Online. How To Geek. © 2024. Dostupné z: https://www.howtogeek.com/devops/what-is-docker-swarm-mode-and-when-should-you-use-it/. [cit. 2024-05-07].

[26] *Nomad vs. Kubernetes*. Online. HashiCorp Developer. © 2024. Dostupné z: https://developer.hashicorp.com/nomad/docs/nomad-vs-kubernetes. [cit. 2024-05-07].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

Příloha P1: Sada vytvořených scriptů pro automatickou tvorbu Kubernetes clusteru uložených na cd přiloženým k práci