VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

PLATFORMA PRO VIRTUALIZACI KOMUNIKAČNÍ INFRASTRUKTURY
COMMUNICATION INFRASTRUCTURE VIRTUALIZATION PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Stodůlka**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Václav Uher, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Tomáš Stodůlka ***ID:*** 174399
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Platforma pro virtualizaci komunikační infrastruktury

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte si problematiku virtualizace infrastruktury se zaměřením na kontejnerovou virtualizaci. Zpracujte podrobný přehled nejznámějších nástrojů pro běh takového prostředí (například Kubernetes, OpenShift, OpenStack) a následně vyberte nejvhodnější nástroj, který splňuje podmínky zadané vedoucím práce. Pro tento nástroj pak podrobně popište filozofii fungování a způsob nasazení. Vybrané prostředí nainstalujte a pro demonstrační účely pak nasaďte ukázkovou aplikaci, která bude obsahovat alespoň dva servery, několik klientů a síť alespoň se třemi podsítěmi včetně nastavení firewallu. Následně změřte nároky daného scénáře na zdroje a výsledky zhodnoťte.

DOPORUČENÁ LITERATURA:

[1] LUKŠA, Marko. Kubernetes in action. Shelter Island, NY: Manning Publications Co., [2018]. ISBN 978-16--7293-726.

[2] BUMGARDNER, V. K. Cody. OpenStack in action. Shelter Island, New York: Manning, 2016. ISBN 978-16--7292-163.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Václav Uher, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práce se zabývá výběrem platformy pro virtualizaci infrastruktury se zaměřením na kontejnerizaci s podporou sandboxingu a následným zkoumáním její náročnosti. Práce začíná vysvětlením základních technologií jako: virtualizace, cloud computing a kontejnerizace spolu s jejími zástupci, kteří danou technologii zprostředkovávají. Zvláštní prostor je vymezen právě pro platformy cloud computingu: Kubernetes, OpenStack a OpenShift. Nejvhodnější platforma je pak vybrána a vlastním způsobem nasazena tak, aby splňovala veškeré podmínky stanovené vedoucím práce. V rámci testování náročnosti vybrané platformy jsou vytvořeny skripty (převážně v jazyce Bash) určené ke skenování vytíženosti systému, vytváření scénářů, simulaci zátěže a automatizaci.

KLÍČOVÁ SLOVA

Virtualizace, virtuální stroj, kontejnerizace, kontejner, sandboxing, cloud computing, Docker, KVM, Kubernetes, OpenStack, OpenShift

ABSTRACT

The thesis deals with selection of infrastructure virtualization platform focusing on containerization with sandboxing support and with following examination of its difficulty. The work begins with an explanation of the basic technologies such as: virtualization, cloud computing and containerization, along with their representatives, that mediate the technology. A special scope is defined for cloud computing platforms: Kubernetes, OpenStack and OpenShift. Futhermore, the most suitable platform is selected and deployed using own technique so that it fullfills all the conditions specified by thesis supervisor. Within the difficulty testing of the selected platform, there are created scripts (mainly in the Bash language) for scanning system load, creating scenarios, stress testing and automation.

KEYWORDS

Virtualization, virtual machine, containerization, container, sandboxing, cloud computing, Docker, KVM, Kubernetes, OpenStack, OpenShift

STODŮLKA, Tomáš. *Platforma pro virtualizaci komunikační infrastruktury*. Brno, 2020, 92 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Václav Uher, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Platforma pro virtualizaci komunikační infrastruktury“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Touto cestou bych velmi rád poděkoval vedoucímu práce Ing. Václavu Uhrovi, Ph.D. za odborné vedení, vstřícnost, konzultace a cenné rady při zpracovávání diplomové práce. Rovněž bych chtěl poděkovat zaměstnancům ze společnosti TietoEVRY, Bc. Oliveru Horečnému a Ing. Ivanu Adamovi za propůjčení části výpočetních serverů. V neposlední řadě bych chtěl poděkovat partnerce Editě Mádrové a nejbližší rodině za velkou podporu během celého studia, zvláště pak bráchovi Bc. Petru Stodůlkovi za korekturu.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Internetové služby a technologie se v dnešní době rapidně vyvíjejí a pro mnohé z nás již představují každodenní nutnost, bez které se v reálném životě neobejdeme. Ať už jde o hledání informací, vzdělávání, sdílení, zábavu, navigaci, ..., či o komunikaci, všichni větší poskytovatelé těchto služeb (Google, Amazon, Facebook, ...) potřebují vysoce výkonné servery/data-centra. Navíc s rostoucí poptávkou roste i nezbytnost neustále zvyšovat jejich výpočetní výkon a kapacitu. Na světě ale snad neexistuje server, který by takovou míru zátěže sám obsloužil, taktéž by bylo nesmyslné jen kvůli zvýšení výkonu kupovat úplně nový, výkonnější stroj. A co teprve v případě, kdy by využití bylo jednorázové, například pro testovací, nebo výzkumné účely?

Částečné řešení přinesla v minulosti virtualizace, která dokázala výkonný server rozdělit na „menší části“, každá pro jednu aplikaci, nastavená přesně dle potřeby. Dnešní náročnosti splňuje technologie, která z virtualizace přímo vychází – cloud computing. Nabízí stejné možnosti a hlavně povyšuje virtualizaci na skupinu vícero zařízení, které mohou pro ostatní vypadat jako jeden celek. Tento celek pak na základě poptávky může vytvářet nové virtuální servery, nebo optimalizovat stávající – přidáním, nebo odebráním výpočetního výkonu a HW (Hardware) zdrojů. Poskytovaný cloud může být **veřejný**, tj. poskytovat správu a výpočetní sílu jiné organizaci přes Internet; **privátní** – vybudován pro vlastní účely v rámci jedné organizace; nebo **hybridní** (kombinací obou). Každý má své výhody i nevýhody, především je důležité rozhodnout, zdali bude z dlouhodobého hlediska výhodnější vysoká počáteční investice, nebo menší, ale přetrvávající platba za poskytované prostředky.

Cílem této práce je nastudovat a seznámit se s problematikou virtualizace infrastruktury se zaměřením na kontejnerovou virtualizaci, především s podporou sandboxingu. Zpracovat přehled nejznámějších platforem: Kubernetes, OpenStack a OpenShift a na základě kritérií zadané vedoucím práce vybrat nejvhodnější z nich. Dále vybranou platformu zprovoznit a otestovat její plnou funkčnost. Následně v ní navrhnout a vytvořit scénáře, na kterých se změří jejich náročnost na HW prostředky v rámci zvolené platformy. Výsledně zvolená platforma a zjištěné výsledky budou představovat základ pro projekt: Kybernetická aréna pro výzkum, testování a edukaci v oblasti kyberbezpečnosti[1].

První kapitola představuje teoretickou část diplomové práce. Ze začátku vysvětluje základní technologie jako: virtualizace, cloud computing a kontejnerizace. Následně je i probrán zástupce, který danou technologii zprostředkovává a je používaný i v následujících platformách. Zvláštní prostor je vymezen pro platformy cloud computingu: Kubernetes, OpenStack a OpenShift. V každé platformě je kla-

[1] Více informací na stránce: `https://starfos.tacr.cz/cs/project/VI20192022132`.

den důraz na její vlastnosti, architekturu, možnosti kontejnerizace s podporou sandboxingu, síťování s podporou tvorby firewallů, instalace a její minimální hardwarové požadavky pro nasazení. Ve výstupu z teoretické části jsou uvedeny požadavky stanovené vedoucím práce. Dále výhody a nevýhody platforem a celkové shrnutí s výběrem platformy, která nejvíce splňuje zadané požadavky.

Druhá kapitola se věnuje přípravě prostředí pro nasazení zvolené platformy a metodikou instalace s testováním funkčnosti jednotlivých komponent.

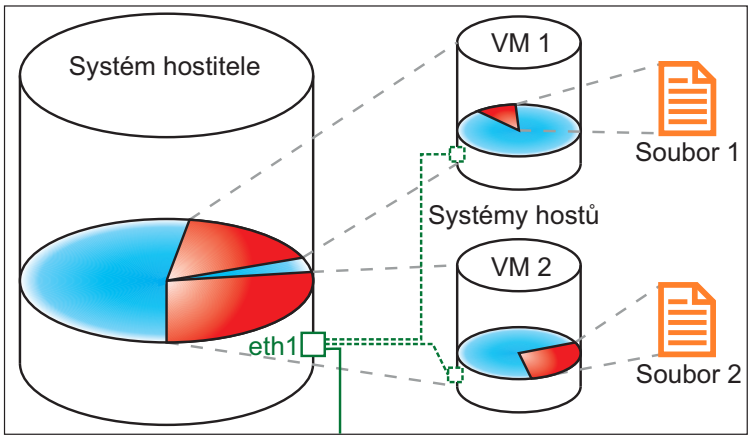
Třetí kapitola se zabývá měřením náročnosti zvolené platformy s následnou prezentací výsledků. Celkově je rozdělena do tří skupin scénářů: první se zaměřením na náročnost dílčích funkcí zvolené platformy; druhá s implementací pokročilejšího scénáře, který systém s nasazenou platformu přetěžuje; a třetí s praktičtější implementací a rozdělením scénáře druhé skupiny do menších celků s cílem zjistit, jak je systém ovlivňován přidáváním virtuální infrastruktury – komunikace v rámci stejné sítě (bez a s web-serverem), dvou sítí přes směrovač, dvou sítí přes firewall. Kapitola je zakončena shrnutím vyhodnocených výsledků.

1 Teoretická část práce

Tato kapitola je zaměřena na vysvětlení základních technologií jako jsou: virtualizace, cloud computing a kontejnerizace. Následně jsou více do hloubky rozebrány platformy cloud computingu: Kubernetes, OpenStack a OpenShift – jejich služby, vlastnosti, architektura, možnosti kontejnerizace, síťování, tvorba firewallu, instalace a minimální požadavky pro nasazení na servery.

1.1 Virtualizace

Virtualizace je technologie, která vytváří virtuální verze z fyzických zdrojů, jako jsou výpočetní hardware, datové uložiště nebo počítačové sítě. V oblasti výpočetní techniky je fyzický systém – na kterém běží virtuální stroje (VMs) – nazýván hostitel (angl. Host). Příkladné použití virtualizace znázorňuje obrázek 1.1: 2 soubory jsou uloženy na dvou odlišných discích v rámci VMs (taktéž označovány jako hosté, angl. Guests), které jsou součástí oddílu na fyzickém disku hostitele. Přitom oba VMs mohou mít svou vlastní síť připojenou k hostiteli, aniž by potřebovaly další dvě síťová rozhraní [1, 2].



Obr. 1.1: Virtualizace systému

Mezi největší výhody virtualizace patří [1, 2]:

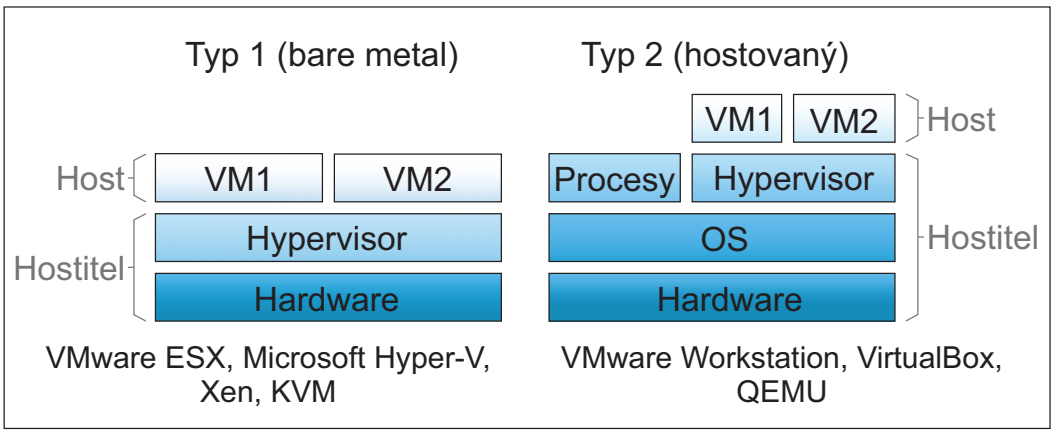
- **Optimalizace zdrojů** – efektivní využívání výpočetní síly, uložiště a přenosové šířky sítě podle potřeb uživatelů a aplikací. Nedojde pak například k situaci, kdy bude použito tří serverů (každý pro 1 aplikaci), ale jejich vytížení CPU se bude na jednom serveru přibližovat sotva k 25 %.
- **Snížení nákladů** – virtualizace snižuje celkový počet fyzických serverů a potřebný prostor k jejich umístění. Menší počet serverů bude generovat menší

množství tepla a tím pádem klesne i potřeba chlazení racků klimatizacemi. Ve finále velkým organizacím s okolo 100–1000 serverů podstatně klesnou náklady za počáteční nasazení i za každodenní spotřebu el. energie.

- **Bezpečnost** – vytvořené VMs běží v tzv. sandboxovém prostředí – nemohou se navzájem narušovat a jejich vykonávané procesy mají omezený přístup ke zdrojům hostitele (včetně jeho sítí a portů). Tato vlastnost může být dále využita ke spouštění, či testování programů z nedůvěryhodných zdrojů, aniž by mohly poškodit hostitele, jeho operační systém (OS), nebo ostatní VMs.
- **Ladění (Debugging) a testování** – umožňuje jednoduše vytvořit kopii běžícího OS pro účely ladění, či testování.

Součástí hostitele je hypervisor (virtualizační software), který se stará právě o vytváření, řízení a správu VMs a přiděluje jim HW prostředky hostitele. V dnešní době existuje mnoho komplexních a stabilních hypervisorů jako jsou například: RedHat KVM, VMware, Microsoft Hyper-V, Oracle VirtualBox a Xen. Hlavním rozdílem mezi nimi je jejich klasifikace (obr. 1.2), která spočívá v umístění hypervisora v systému [1, 3]:

- **Hypervisor typu 1** – běží přímo na fyzickém HW, který dokáže řídit stejně tak jako VMs. Je taktéž označován jako hypervisor „bare metal“.
- **Hypervisor typu 2** – běží jako aplikace v operačním systému, která je nainstalovaná na fyzickém HW. Je zpravidla pomalejší, protože komunikace VMs musí procházet přes OS hostitele, předtím než dorazí k HW, ale zato umožňuje navíc i spoustu vlastních přizpůsobení.



Obr. 1.2: Klasifikační třídy hypervisoru

1.1.1 KVM-QEMU

Zvláštním případem může být hypervisor KVM-QEMU, který se instaluje na úrovni operačního systému. Teoreticky by měl spadat pod kategorii hypervisora typu 2, avšak celkově se chová jako hypervisor typu 1. KVM (Kernel-based Virtual Machine) je sada modulů jádra pro HW architektury x86 s virtualizačními rozšířeními a při načtení převede jádro Linuxu do hypervizora. Načítací moduly jsou *kvm.ko* (poskytuje podstatné možnosti virtualizace) a *kvm-intel.ko*, nebo *kvm-amd.ko* (záleží na specifikaci procesoru) [1, 3].

Spolu s KVM je instalován emulátor QEMU-KVM pro emulaci HW periferií. V podstatě samotný KVM nestačí pro běh VMs a naopak samotný QEMU je příliš pomalý – při správě HW prostředků musí používat dynamické překlady, jelikož vše vyřizuje přes OS hostitele. Proto byl QEMU modifikován do QEMU-KVM, který může vzájemně komunikovat přímo s KVM moduly a tedy i vykonávat instrukce VM přímo na CPU (bez dříve potřebných dynamických překladů) [3].

V poslední řádě je zapotřebí libvirt – sada nástrojů pro řízení mnoha aplikací a hypervisorů, mezi které patří i QEMU-KVM. Avšak toto řízení nemusí být jen v rámci jednoho fyzického serveru, protože libvirt umožňuje spravovat QEMU-KVM i vzdáleně [3]. Tato vlastnost je velmi populární v oblasti cloud computingu, viz následující kapitola 1.2.

1.2 Cloud computing

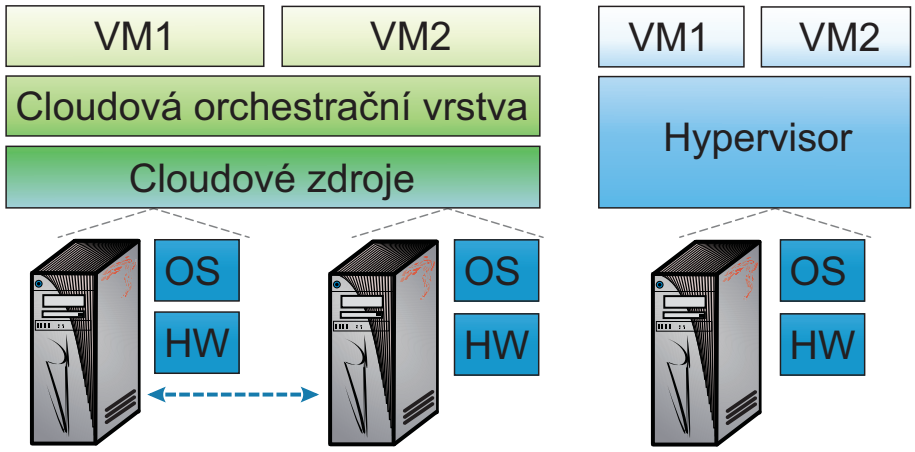
Cloud computing povyšuje virtualizaci na úroveň skupiny fyzických zařízení, které mezi sebou komunikují přes vysokorychlostní síť. Rozdíl mezi obyčejnou virtualizací a cloudovým systémem je patrný z obrázku 1.3. Každé zařízení připojené do cloudového systému, virtualizuje své HW prostředky do cloudů[1], které jsou následně orchestrovány[2] cloudovým managementem[3] a automatizačním softwarem [1, 2]. Cloudové zdroje pak mohou být použity přesně dle aktuálních potřeb uživatelů a sloužit k nejrůznějším účelům vytvářením VMs, počítačových sítí, kontejnerů (kap. 1.3) atd.

Cloud computing klade důraz na vysokou bezpečnost, spolehlivost, škálovatelnost, dostupnost a mnoho dalšího. Například softwarová migrace – VM může být přenesen na další server a to i dokonce, aniž by byla ovlivněna jeho funkčnost (např. přístup k databázi) – taktéž nazývaná jako živá migrace (angl. Live Migration) [1].

[1] Množiny (angl. Pools) virtuálních zdrojů.

[2] Pojem **orchestrace** představuje uspořádání automatizačních procesů tak, aby sloužily konkrétním cílům. Orchestrace pomáhá automatizovat celé sbírky úkolů, zatímco automatizace řeší typicky jen jeden úkol [4].

[3] Software pro správu a monitorování všeho co funguje v cloudu.



Obr. 1.3: Porovnání cloud computingu (vlevo) a virtualizace (vpravo)

Cloud computing je implementován pomocí SW balíčků a mezi ty nejznámější patří Kubernetes (kap. 1.4), OpenStack (kap. 1.5), či OpenShift (kap. 1.6). Každá platforma používá ve své dokumentaci trochu odlišné výrazy, které ovšem plní stejný účel. Proto budou k synchronizaci použity následující:

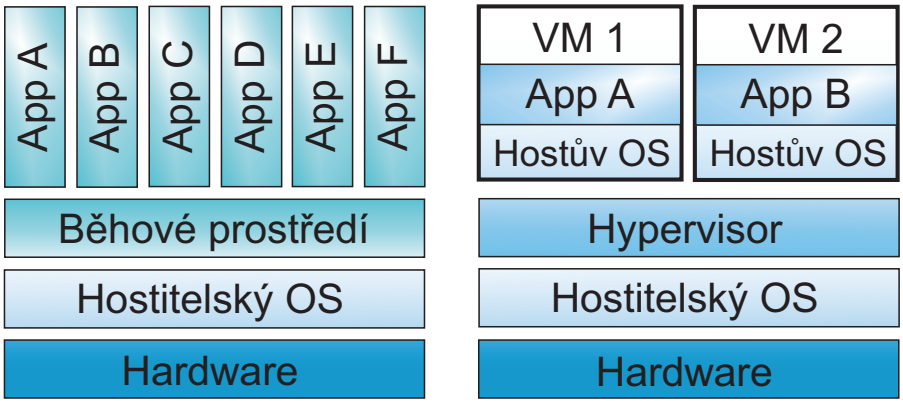
- **Kontrolér** (Controller Node) – zařízení řídící výpočetní uzly v cloudu.
- **Výpočetní uzel** (Compute Node) – zařízení vykonávající přiřazené úkoly od kontroléru, poskytuje své HW zdroje kontroléru/cloudu.
- **Tenant** – skupina vlastnící a spravující zdroje v rámci jednoho projektu.
- **Tenant síť** – skupina sítí v rámci jednoho projektu.

Při použití více než jednoho kontroléru zajišťuje cloud computing vysokou dostupnost (HA), kdy v případě selhání jednoho kontroléru převezme činnost jiný a služby poskytované na výpočetních uzlech tedy nebudou omezeny.

1.3 Kontejnerizace

Kontejnerizace (taktéž tzv. „odlehčená“ virtualizace) je další úrovní virtualizace, avšak na aplikační úrovni operačního systému. V kontejneru již neběží jeho vlastní OS, místo toho využívá stejné jádro operačního systému a balíčky aplikací hostitele spolu s dalšími kontejnery. Nejpodstatnější rozdíl mezi VMs a kontejnery specifikuje obrázek 1.4. Kontejnery namísto hardwaru virtualizují operační systém, spotřebovávají méně HW zdrojů než VMs a tedy i zvládají běh více aplikací [2, 5]. Běhové prostředí kontejneru (angl. Container Runtime) představuje SW, nebo nástroj, který dokáže vytvářet a spravovat kontejnery. V závislosti na možnostech takového nástroje se běhová prostředí rozdělují na [6]:

- **nízkoúrovňová** (angl. Low-level) – odpovědné pouze za spuštění kontejneru (např. `runc`),
- a **vysokoúrovňová** (angl. High-level) – pro vytváření kontejnerů využívají nízkoúrovňové běhové prostředí a navíc podporují správu kontejnerových obrazů, API, ..., registry, (např. `containerd`, který využívá aplikace Docker, více v sekci 1.3.1).



Obr. 1.4: Rozdíl mezi kontejnerizací (vlevo) a virtualizací (vpravo)

K vytvoření kontejneru je zapotřebí dvou částí [7]:

- diskový obraz kontejneru (angl. Container Image),
- mechanismy operačního systému, které izolují běžící procesy:
  - **jmenný prostor** (angl. Namespace) – odděluje systémové zdroje jako jsou: souborový systém, ID procesů, ID uživatelů, síťová rozhraní, atd.,
  - **Kontrolní skupiny** (angl. Control Groups, `cgroups`) – omezují množství zdrojů, které může spotřebovat (CPU, RAM, ..., přenosová rychlost).

Diskový obraz kontejneru obsahuje tzv. „aplikační běhové prostředí“, které obsahuje binární soubory, knihovny a další data potřebná k běhu kontejneru. Do obrazu může být zabalena i aplikace s různými příkazy a soubory pro její správné spuštění. Takto vytvořený obraz je velmi jednoduše přenositelný mezi systémy. Aplikace v takovém kontejneru budou mít totožné chování i v případech, kdy mají systémy rozdílná nastavení. Nutnou podmínkou je ale kompatibilita s jádrem systému, pro který byl obraz vytvořen. Například obraz použitý v kontejneru na Linuxové distribuci Ubuntu, je možné přenést a použít na dalších distribucích Linuxu (CentOS, RHEL, ..., Fedora), avšak ne na jiných OS jako je Windows či macOS [1].

Zpravidla každý Linuxový systém má zpočátku jeden jmenný prostor obsahující všechny systémové zdroje, které se mohou následně dělit použitím dalšího jmenného prostoru. Proces uvnitř jmenného prostoru vidí pouze zdroje, které jsou ve stejném jmenném prostoru a spolu s `cgroups` nemohou využít více HW prostředků, než mu

bylo přiděleno. Takto jsou aplikace uvnitř kontejneru izolovány i se vším potřebným k jejich fungování (systémové nástroje, knihovny, nastavení prostředí, atd.), od zbytku OS a dalších kontejnerů. Avšak tento způsob izolace není zdaleka tak silný, jako je tomu u virtualizace, protože samotné kontejnery standardně nepodporují sandboxing. Naopak hlavní výhody kontejnerizace jsou nižší režijní náklady a efektivnější využití HW zdrojů. Kontejner využívá výpočetní zdroje dynamicky a po provedení potřebných operací je může hostitel opět použít pro další účely [7].

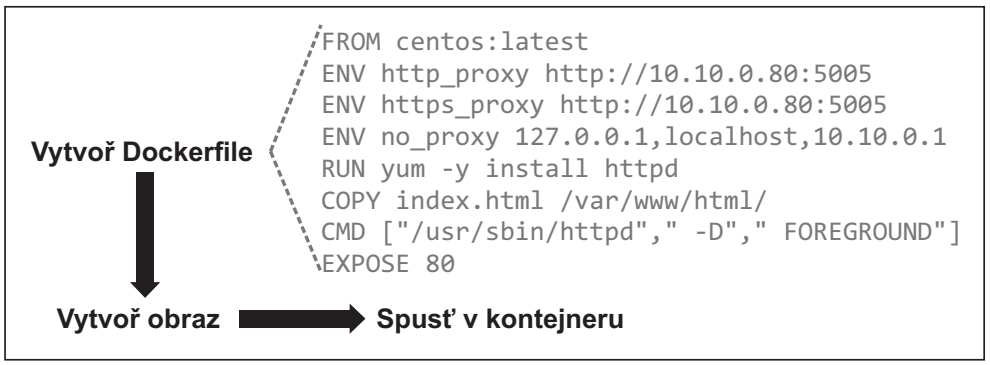
Kontejner používá svůj vlastní souborový systém, který se ale ukládá do kontejnerové vrstvy. To hlavně znamená, že uložiště pro kontejnery nejsou perzistentní a po jeho restartu veškerá data aplikací v rámci kontejneru zanikají. Pro uchovávání dat se ke kontejneru připojují většinou volumy[4] (například metodou LVM – angl. Logical Volume Management), nebo „bind mounty“ [8].

1.3.1 Docker

Dle průzkumu [9] je Docker nejvíce rozšířený open-source SW určený ke kontejnerizaci mezi organizacemi. Vychází z něj celá řada dalších technologií jako jsou: Kubernetes, Docker compose, Docker Swarm a Apache Mesos. Vytváření kontejneru je zpravidla několikrát rychlejší než u VM a navíc jeho diskový obraz zabírá podstatně méně místa na disku hostitele. Každý obraz Dockeru je tvořen řadou vrstev a s každou další změnou je vytvořena nová vrstva. Další obraz Dockeru je seskládán z vrstev předešlého obrazu a místo na disku bude zabírat pouze jejich skutečný rozdíl proti předchozí vrstvě. Vrstvení obrazu má i svou další výhodu – umožňuje vrátit obraz do předešlého stavu (například po kroku či aktualizaci vedoucí k chybě), angl. „Rollback“ [10].

Je možné stáhnout již vytvořený obraz kontejneru z veřejných registrů (např. Docker Hub). Následně jej lze modifikovat dle vlastních potřeb a vytvořit nový obraz definováním souboru **Dockerfile**. Každá instrukce v souboru Dockerfile vytváří novou vrstvu v obrazu. Obrázek 1.5 zobrazuje příkladný postup při nasazení vlastního obrazu pro spuštění webové aplikace. V prvním kroku je vytvořen Dockerfile, který jako základní obraz používá centOS. Následně nastavuje proxy prostředí, nainstaluje službu `httpd`, zkopíruje konfigurační soubor, spustí `httpd` na popředí a nakonec bude naslouchat na portu 80. Příkazem `docker build` vytvoří obraz, který je možné spustit v kontejneru.

[4] Pojem volume (česky svazek) bývá často zaměňován s diskovými oddíly, ve skutečnosti se ale od sebe liší. Pevný disk, může být rozdělen na více oddílu, které mohou (ale i nemusí) být naformátované a rovněž nemusí mít nutně souborový systém a tím pádem být pro některé systémy viditelné (např. ve Windows). Volumy jsou pak vytvářeny z těchto oddílů, nebo rovnou celých disků (např. DVD, CD) a vždy obsahují souborový systém. Navíc může volume skládat více oddílů do jednoho tzv. logického volumu.



Obr. 1.5: Vytvoření Docker obrazu pomocí souboru Dockerfile

Vytvořené obrazy je možné uchovávat vzdáleně na Docker Hubu, ze kterého lze obrazy stahovat pohodlně na další zařízení. Avšak nevýhodou je, že verze zdarma umožňuje vlastnit pouze 1 repositář s privátním přístupem. Jako řešení přichází lokální Docker registr ve dvou variantách:

- **Nezabezpečený** – kdokoliv v lokální síti může přistoupit do vytvořeného Docker registru pokud zná IP serveru.
- **Zabezpečený** – pro registr je vygenerován privátní klíč a certifikát s jeho kanonickým jménem. Oba tyto soubory jsou následně zakomponovány do kontejneru s běžícím registrem. Přístup do registru má pak pouze zařízení se stejným certifikátem.

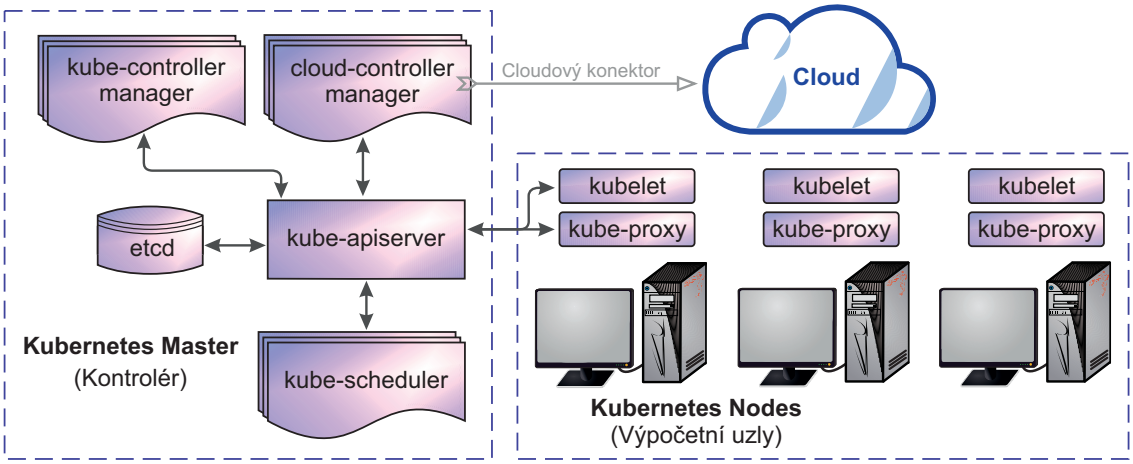
1.4 Kubernetes

Kubernetes, nebo-li k8s (k+„8 znaků“+s) je open-source platforma orchestrující nasazení kontejnerizovaných aplikací. Patří do tzv. skupiny COEs – Container Orchestration Engines. Původně jej navrhla a vyvíjela společnost Google, která jej otevřela v roce 2014 veřejnosti. Později se do projektu zapojila i společnost RedHat a další [11]. Tyto dvě firmy představují na základě [12] největší přispěvatele do vývoje k8s – s 41,3% podílem od Googlu a 16% podílem od RedHatu. V dokumentaci ke Kubernetes se používají následující výrazy [11]:

- **Cluster** – uskupení více zařízení, které pracují společně.
- **Master** – plní funkci kontroléru.
- **Node** (taktéž označovaný jako Worker) – vykonává funkci výpočetního uzlu.
- **Pod** – skupina jednoho, nebo více kontejnerů, které jsou nasazené na stejném výpočetním uzlu. Je taktéž nejnižší nasazující jednotkou v k8s.

1.4.1 Architektura

Kubernetes je kolekcí různých aplikací, které spolupracují společně aniž by o sobě celkově věděly. Výhodou takového nasazení je flexibilita – část nějaké funkce může být vyměněna bez toho aniž by si zbytek systému něčeho všiml. Nevýhodou je zase složitost, protože nasazení, monitorování a porozumění takovému systému vyžaduje více znalostí a konfigurací různých nástrojů [7].



Obr. 1.6: Architektura platformy Kubernetes

Z fyzického pohledu je k8s systém, který uskupuje více zařízení do jediné jednotky (clusteru), ve které se rozdělují zařízení na dvě skupiny: kontroléry a výpočetní uzly. V jednom clusteru by měl být minimálně jeden kontrolér a jeden výpočetní uzel. Nasazení k8s clusteru spolu s jeho jednotlivými komponenty zobrazuje obrázek 1.6 [11]. Mezi komponenty kontroléru patří [7, 11]:

- **Etcd** – je srdcem k8s clusteru. Představuje vysoce dostupné uložiště typu klíč-hodnota, které uchovává celou konfiguraci clusteru.
- **Kube-apiserver** – je jediným serverem, který má přímý přístup do k8s clusteru. Api server zprostředkovává veškerou interakci mezi ostatními uživateli, k8s uzly a API objekty uloženými v etcd.
- **Kube-scheduler** – skenuje API server ohledně nových vytvořených podů, které ještě nemají přiřazený žádný výpočetní uzel. Následně určí nejlepší výpočetní uzel, který by ho mohl spustit.
- **Controller-manager** – zavádí do k8s řídící smyčky, které sledují a reagují na události či požadavky v clusteru. Pro běh podů není povinný, nicméně obohacuje k8s o mnoho užitečných funkcí v rámci:
  - **Kube** (KCM) – sleduje a reaguje na události v případě, že jeden vý-

početní uzel, nebo pod, selže. Dále umožňuje funkce jako ReplicaSet[5] a Deployments[6].

  - **Cloud** (CCM) – běží na kontrolérech, které spolupracují s poskytovateli cloudových služeb a částečně přebírá funkce KCM. Příkladem takového poskytovatele může být například OpenStack CCM.

Komponenty kontroléru udržují a řídí stav clusteru, ale nespouštějí žádné kontejnerizované aplikace. K tomu slouží komponenty výpočetních uzlů [7, 11]:

- **Kubelet** – je mostem, který spojuje dostupné HW zdroje (procesor, disk a paměť) výpočetního uzlu s k8s clusterem. Komunikuje s API serverem a vytváří kontejnery v podech, které mají běžet na daném uzlu. Je také zodpovědný za sledování stavu běžících kontejnerů v podech a v případě potřeby jej automaticky restartuje a ohlašuje na API server.
- **Kube-proxy** – je odpovědný za implementaci k8s služby[7]. Stará se o síťová pravidla na daném uzlu, přiřazuje službám virtuální IP (VIP) adresu a implementuje vyrovnávání zátěže.
- **Běhové prostředí kontejnerů** – například Docker.

Kromě základních komponent výše existuje i celá řada dalších, doplňkových, které přidávají do k8s nové vlastnosti – mohou jimi být například CoreDNS, Dashboard (webové grafické uživatelské rozhraní – GUI), různá síťová rozšíření (viz kap. 1.4.3) a mnoho dalších na oficiálních stránkách Kubernetes[8] [11].

1.4.2 Kontejnerizace

Jak už bylo zmíněno dříve, k8s se specializuje především na orchestraci kontejnerů. Obecný postup zobrazuje nasazení aplikace na obrázku 1.7. Administrátor vytvoří obrazy kontejnerů obsahující komponenty aplikace, nahraje je na privátní registr (například: Google Container Registry, Amazon Elastic Container Registry, ..., Docker Hub) a následně zašle popis aplikace[9] na API server. V tomto případě popis aplikace obsahuje 4 kontejnery seskupené ve třech podech. První 2 pody obsahují pouze 1 kontejner, zatímco třetí obsahuje 2 kontejnery – to znamená, že tyto 2 kontejnery musí být umístěny na stejném výpočetním uzlu a nesmí být od sebe zcela izolovány. Kontejnerové běhové prostředí se postará o to, aby každý kontejner v rámci jednoho

[5] Definuje kolik replik podů bude vytvořeno. Více podů může být použito k rozložení zátěže, nebo k ochraně služby při selhání.

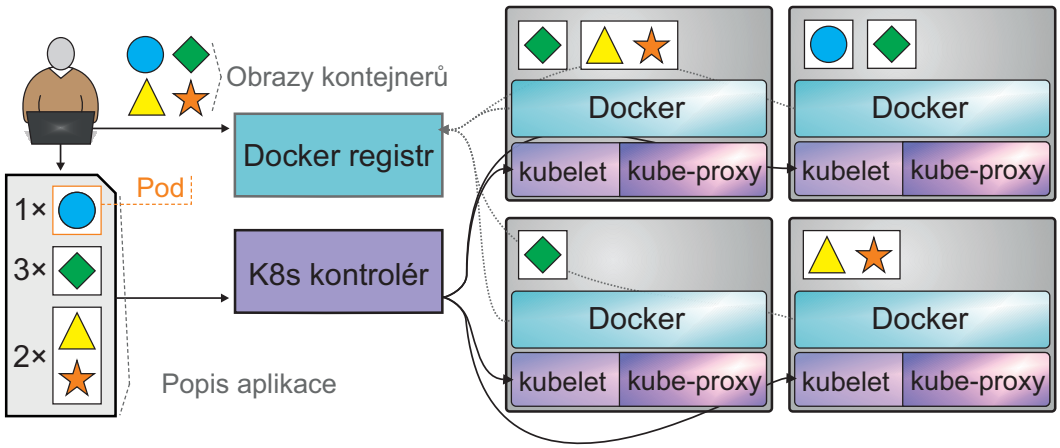
[6] Poskytuje aktualizace pro pody a repliky – Rollout, Rollback, atd.

[7] Abstraktní způsob, jak nasadit aplikaci na více podech jako síťovou službu.

[8] `https://kubernetes.io/docs/concepts/cluster-administration/addons/`

[9] Instrukce pro k8s: jak nasadit obrazy do kontejnerů, jaký mají mít mezi sebou vztah a které kontejnery je potřeba spustit společně na jednom uzlu. Pro každý pod je možné specifikovat i počet replik (kopií) nebo přiřadit IP adresu a učinit jej dostupným pro ostatní pody, či pro interní, nebo externí sítě.

podu využíval stejnou sadu jmenných prostorů, s výjimkou souborového systému – ten může být sdílen pouze skrz připojený volume. API server dále zpracovává požadavek a zasílá jej na kube-scheduler, který naplánuje nasazení podů (dle počtu replik) na dostupné výpočetní uzly. Výběr výpočetního uzlu je zvolen v závislosti na HW požadavcích podů a taktéž na volných výpočetních zdrojích jednotlivých uzlů. Kubelet na zvolených uzlech předává instrukce běhovému prostředí (v tomto případě Dockeru), aby stáhl požadované obrazy kontejnerů z registru a spustil je v kontejneru/podu [14].



Obr. 1.7: Obecné schéma spuštění kontejnerizované aplikace v k8s

K8s po nasazení aplikace kontroluje stav podů, zdali jejich počet stále odpovídá přesně popisu aplikace. Pokud se nějaký pod zasekne, nebo skončí chybou, k8s jej automaticky restartuje. Taktéž pokud přestane fungovat celý výpočetní uzel, k8s automaticky zajistí vytvoření zasažených podů na novém uzlu [14].

V případě potřeby je možné i změnit popis aplikace – například přidat, nebo ubrat počet replik. K8s pak automaticky nasadí další pody, nebo stopne stávající. Poslední možností je nechat rozhodnutí na k8s, který může optimálně přidávat pody v závislosti na aktuálním vytížení CPU, spotřeby RAM, nebo jakékoliv další metriky potřebné pro nasazenou aplikaci [14].

Sandboxing

Pody nabízejí k8s vyšší izolaci než samotné kontejnery, podmínky sandboxingu ale stále nesplňují. Na výpočetních uzlech je potřeba nainstalovat a povolit jednu z aplikací:

- **gVisor** – implementuje mezi hostitele a kontejner další vrstvu jádra Linuxu. Zahrnuje nové OCI (vysvětleno v sekci **Komunikace mezi pody a k8s**) běhové prostředí `runsc` [15].

- **GKE sandbox** – vychází z projektu gVisor, kde převážně zjednodušuje jeho používání pro pody v k8s. Při vytváření podu by pak měla být k dispozici nová možnost, zvolit GKE sandbox [16].

- **Kata kontejnery** – pouze v případě, kdy k8s používá běhové prostředí `containerd` s podporou CRI (vysvětleno v sekci **Komunikace mezi pody a k8s**). Projekt Kata kontejnerů je více rozebrán v sekci **Sandboxing** OpenStacku.

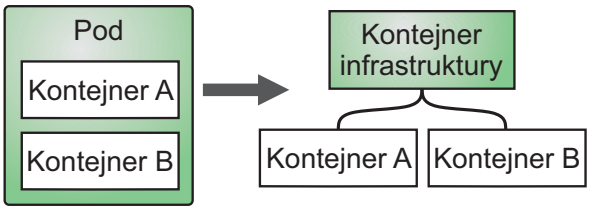
1.4.3 Síťování

Jelikož se počty podů mohou různě měnit – stejně tak jako jejich samotné umístění v rámci celého clusteru – zaujímá síťování v k8s velmi významnou roli, ať už jde o komunikaci mezi kontejnery, pody, či klienty, kteří se chtějí připojit ke kontejnerizované aplikaci.

Komunikace mezi kontejnery v rámci stejného podu

Kontejnery v rámci jednoho podu běží ve stejném síťovém jmenném prostoru a sdílí stejnou přiřazenou IP adresu a port. Pro běžící kontejnery to znamená, že nevyužívají stejný port, jelikož se tak mohou dostat do konfliktu. Tato situace se týká pouze stejného podu, kontejnery v dalších podech využívají své vlastní jmenné prostory. Všechny kontejnery uvnitř podu mají taktéž stejné loopback rozhraní, přes které mohou komunikovat s ostatními kontejnery [14].

Síťový jmenný prostor podu ve skutečnosti tvoří další kontejner, který představuje infrastrukturu podu (obr. 1.8). Tento kontejner se vytváří jako první (stejně tak i zaniká jako poslední v životním cyklu podu) a po zajištění potřebného síťování se změní do zastaveného stavu – tedy výpočetně neprovádí žádnou činnost. Následuje vytvoření kontejnerů aplikace, které již používají kontejner infrastruktury jako společný jmenný prostor [7, 14].

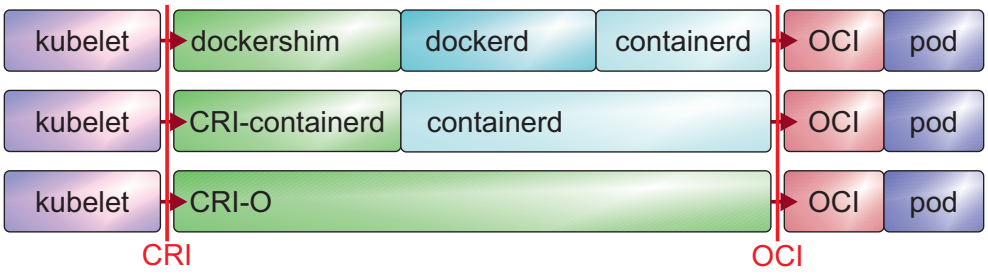


Obr. 1.8: Síťová infrastruktura v rámci jednoho podu

Komunikace mezi pody a k8s

K této problematice se úzce vztahují pojmy **CRI** (Container Runtime Interface) a **OCI** (Open Container Initiative). OCI specifikuje formát kontejneru (běhové prostředí: `runtime-spec` a obraz kontejneru: `image-spec`). Zpočátku k8s pracoval pouze s jednou kontejnerizační technologií – Dockerem. Docker daemon navíc ve výchozím stavu zpracovává jen kontejnery na jednom hostiteli, a tak k8s spolu s Googlem vynalezli „fat daemony“ k orchestraci a řízení Dockeru napříč stovkami uzlů. Později s touto znalostí definovala k8s komunita CRI – rozhraní, umožňující komunikaci mezi různými OCI a komponentou `kubelet`, [13].

Ve výsledku k8s podporuje různá běhová prostředí bez potřeby zasahovat do jeho kódu. Například Docker, CRI-O (využívá OpenShift, ale lze doinstalovat i do k8s), containerd a Frakti. První 3 příklady zobrazuje obrázek 1.9 [6].



Obr. 1.9: Příklady komunikace k8s s pody přes Docker, containerd nebo CRI-O

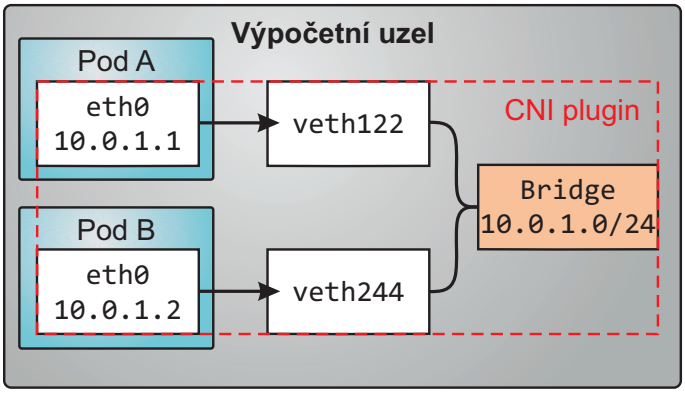
Komunikace mezi pody

Ať už pody běží v rámci stejného výpočetního uzlu či ne, mělo by být všem podům umožněno komunikovat mezi sebou. O samotné zprostředkování spojení se však nestará samotný k8s, nýbrž systémový **administrátor** nebo **CNI** (Container Network Interface) **plugin** – doplňkový komponent. Síť, kterou pody využívají ke komunikaci musí být taková, aby IP adresa pro daný pod byla z pohledu dalších podů úplně stejná – tedy síť bez natovaného[10] provozu. Technika NAT se uplatňuje pouze v případě, kdy pody komunikují s externími službami (do internetu), jelikož IP adresy podů jsou privátní. Zároveň IP adresa pro každý pod musí být unikátní [7, 14].

Situaci v rámci jednoho výpočentího uzlu zobrazuje obrázek 1.10. Síťové rozhraní podu je tvořeno kontejnerem infrastruktury, ale ještě před jeho startem je pro něj vytvořen pár virtuálního rozhraní `veth` (Virtual ETHernet). První rozhraní `veth`***xxx*** přiléhá jmennému prostoru hostitele, zatímco druhé je přesunuto do jmenného prostoru kontejneru infrastruktury a přejmenováno na `eth0`. Rozhraní `veth`***xxx*** je pak

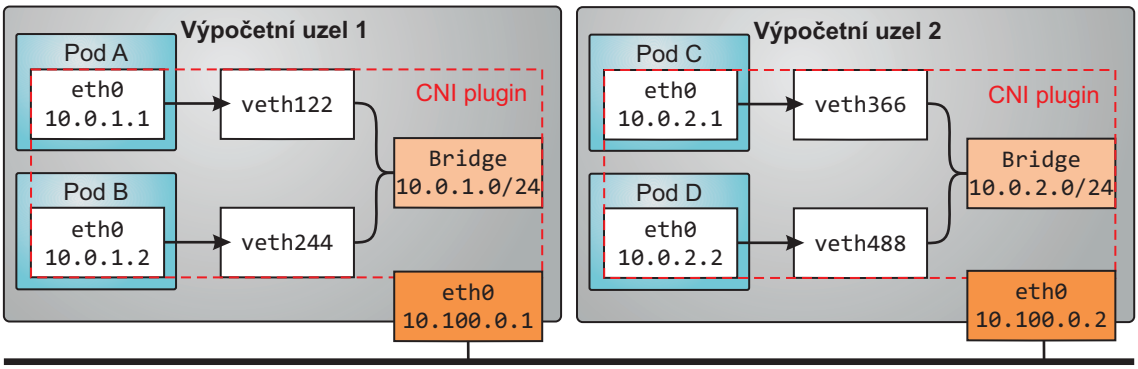
[10]NAT (Network Address Translation) – mapování privátních IP adres na veřejné.

připojeno k síťovému mostu (angl. Bridge), který je nakonfigurován příslušným běhovým prostředím uzlu. Rozhraní `eth0` v kontejneru získá IP adresu z adresního prostoru síťového mostu. Následně mohou pody A i B mezi sebou bez problému komunikovat [14].



Obr. 1.10: Síťová infrastruktura mezi pody v rámci jednoho výpočetního uzlu

Obdobně jsou pody připojeny do sítě v dalších uzlech, viz obr. 1.11. V takovém případě musí být zajištěno, aby každý pod měl svoji unikátní IP adresu, proto síťové mosty využívají různých adresových prostorů. Komunikace mezi výpočetními uzly je pak řešena běžným L3 směrováním – přidáním dvou statických směrovacích cest. Například na uzlu 1: všechny pakety s cílovou IP v rozsahu 10.0.2.0/24 jsou směrovány na 10.100.0.2 [14].



Obr. 1.11: Síťová infrastruktura mezi pody v rámci více výpočetních uzlů

Jak již bylo zmíněno, síťování mezi pody zajišťuje převážně CNI plugin, který celkově zjednodušuje jinak komplikovanou konfiguraci. Před výběrem CNI pluginu je důležité si ujasnit jaké funkce by měl splňovat a k budování jaké topologie bude určen [7]. Například v rámci této diplomové práce je požadováno, aby platforma

byla schopna připojovat kontejnery do různých sítí a řídit mezi nimi provoz pomocí směrovačů a firewallů – to vše na úrovni operačního systému.

Funkce firewallu může být v k8s přirovnána k pojmu tzv. síťové politiky, které jsou aplikovány na skupinu podů dle přiřazeného názvu selektoru. Pravidla síťové politiky mohou specifikovat, jaký provoz bude povolen z/do podu, jmenného prostoru nebo IP rozsahu na základě protokolů (TCP, UDP, SCTP), jmen portů nebo čísel portů [11, 14].

Všechny CNI pluginy pro k8s jsou dostupné na stránkách kubernetes.io[11], z nichž by pro naše požadavky mohly být zajímavé následující [11]:

- **Kube-router** – vytváří jednoduché síťování mezi pody bez použití překryvných sítí VXLAN[12]. Nasazuje na každý výpočetní uzel virtuální směrovač, který řídí provoz mezi pody de-facto jako na obrázku 1.11. V rámci více uzlů je použito BGP k výměně směrovacích údajů. Umožňuje na virtuální směrovače nastavovat síťovou politiku [18].
- **Calico** – poskytuje síťování na L3 vrstvě ISO/OSI a také zavádění bezpečnostních síťových politik. Na každém uzlu se spouští Calico daemon nazývaný Felix, který je zodpovědný za programování směrovacích cest a přístupových seznamů (ACLs). ACLs dále přiřazuje ke zvoleným portům. Calico taktéž podporuje BGP protokol pro distribuci směrovacích informací do dalších sítí, nebo vzájemné propojení s dalším k8s clusterem. V opačném případě může využívat zapouzdřování přenosu do VXLAN překryvných sítí bez potřeb BGP protokolu [19].
- **Contiv** – poskytuje konfigurovatelné síťování (L3 pomocí BGP; překryvné používáním VXLAN; L2 pomocí VLAN; Cisco ACI – centrální aplikační infrastruktura) a rozšířené možnosti nastavení bezpečnostní politiky, která dokáže omezovat i maximální využití přenosové rychlosti v síti pro specifickou skupinu kontejnerů nebo dokonce i virtuálních strojů [20].
- **Flannel** – zavádí do k8s metodu velmi jednoduchých překryvných sítí. Neumožňuje nastavovat síťovou politiku, pro tuto možnost jej lze nasadit v kombinaci s projektem Calico.
- **Romana** – směrování na L3 pomocí BGP, OSPF a na L2 pomocí VLAN, také podporuje API pro přidávání síťové politiky.
- **Weave Net** – vytváří překryvnou virtuální síť VXLAN, která umožňuje propojování Docker kontejnerů napříč dalšími uzly, či datovými centry. Vyhledá-

[11] `https://kubernetes.io/docs/concepts/cluster-administration/networking/` v sekci: How to implement the Kubernetes networking model.

[12] Technika určená k provozu virtualizovaných L2 sítí, které jsou propojeny nad L3 sítí. Přenos je řešen zapouzdřením L2 rámce do UDP paketu a v cíli je opět rozbalen. Kvůli přidání dvou hlaviček navíc, je pro VXLAN síť snížena hodnota MTU o 50 B oproti fyzické L3 síti [17].

vání služeb kontejnerů řeší „micro DNS“ umístěné na každém uzlu. Weave net podporuje i síťovou politiku [21].
- **Canal** – sjednocuje Flannel a Calico.
- **Multus** – umožňuje podu vytvářet více rozhraní (standardně může mít pod kromě loopbacku jen jedno rozhraní), které mohou být připojeny k dalším sítím s podporou všech CNI pluginů (jako např. Calico, Contiv, Flannel).

Kube-proxy

V okamžiku nasazování služby je jejímu podu a replikám staticky přiřazena stejná VIP adresa, následně každý kube-proxy učiní danou službu adresovatelnou v uzlu, na kterém běží, pomocí pravidel ve stavovém firewallu `iptables`. Pokud nyní vezmeme v úvahu, že pody B,C,D na obrázku 1.11 představují službu s VIP 172.16.0.1 na portu 80 a pod A chce s touto službou navázat spojení – pakety s cílovou VIP služby jsou předány kernelu OS daného uzlu. Kernel před odesláním zkontroluje pravidla v iptables. V tomto případě najde shodu a cílová adresa s portem jsou přepsány na IP a port náhodně zvoleného cílového podu (B,C, nebo D), nebo pomocí algoritmu vyvažování zátěže (např. Round Robin). Statická VIP může být taktéž dosažitelná pomocí DNS přidáním další k8s komponenty [14].

1.4.4 Instalace k8s s minimálními požadavky

K8s nabízí různá řešení jeho nasazení a nastavení pro cílová prostředí. Ať už pro lokální počítač, cluster vlastních serverů, virtuální stroje běžící v cloudu poskytovatele (Google Compute Engine, Amazon EC2, Microsoft Azure, atd.), nebo v poslední řadě k8s cluster řízený poskytovatelem s podporou minimálních operací (Google Kubernetes Engine, DigitalOcean, atd.). Pro začátečníky nabízí k8s i dvě řešení představující jeho odlehčenou verzi – **Minikube** (nasazení clusteru na jediném uzlu uvnitř VM) a **Kind** (Kubernetes IN Docker – uzly clusteru nasazené v kontejnerech) [14, 22].

Na základě cílového prostředí je vybrán instalační nástroj, od kterého se dále odvíjejí minimální systémové a HW požadavky. Manuální nasazení vlastního clusteru pomocí nástroje **Kubeadm** například požaduje [22]:

- zařízení s OS linux:
  - Ubuntu 16.04+,
  - Debian 9+,
  - CentOS 7,
  - Red Hat Enterprise Linux (RHEL) 7,
  - Fedora 25+,
  - HypriotOS v1.0.1+,

  - Container Linux 1800.6.0+,
- RAM o velikosti alespoň 2 GB na každém zařízení,
- dvě nebo více CPU jader na každém zařízení,
- zapojení všech zařízení do společné sítě,
- unikátní jména hostitelských zařízení, MAC adresy, `product-uuid`,
- volné specifické porty pro k8s komponenty.

Dalším příkladem může být instalační nástroj **Kubespray**, který nasazuje k8s přímo s jeho doplňkovými komponenty pomocí Ansible[13] playbooků. Celkově je potřeba splnit požadavky před spuštěním Ansible playbooků (např. nastavení konfiguračního souboru, souboru Ansible inventáře, SSH konektivity pomocí veřejného klíče na všechny cílové uzly, ...) a mít připravené velikosti RAM pro kontroléry 1500+ MB a výpočetní uzly 1024+ MB [22].

Daleko vyšší požadavek má nástroj **KRIB** pro nasazení k8s clusteru na servery bare metal. Instalace je zde doporučena na Linuxové systémy s minimálně 16 GB RAM [22].

1.5 OpenStack

OpenStack je open-source platforma pro vytváření vysoce škálovatelných privátních, veřejných i hybridních cloudů, které spravují a řídí množiny výpočetních, úložných a síťových zdrojů [24]. OpenStack je složen z kolekcí projektů (každý se specifickým účelem) a jeho první oficiální verzi vydaly v roce 2010 organizace NASA a Rackspace spolu s 25 dalšími společnostmi [25]. Za posledních 10 let bylo vydáno 21 verzí OpenStacku (pojmenovány abecedně, průměrně dvakrát ročně), na kterých se podílelo celkem 488 společností a komunita [26]. Poslední dvě stabilní verze vyšly v:

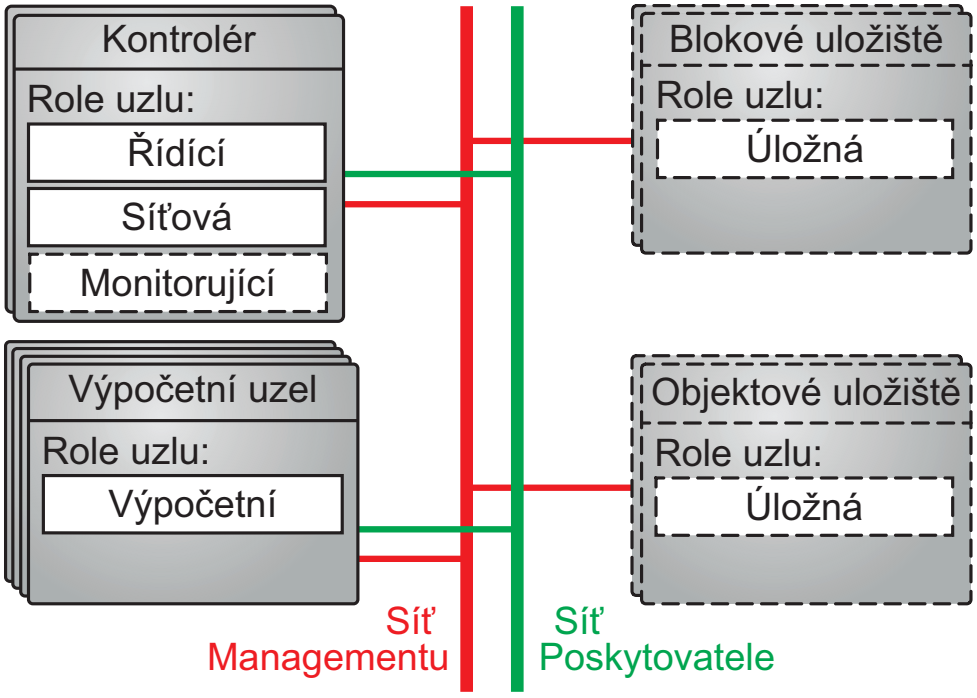
- říjnu 2019 s názvem **Train**
- a květnu 2020 s názvem **Ussuri**.

Aktuálně dle počtu commitů v [26] představuje největších 5 přispěvatelů OpenStacku: RedHat (17,9 %), Mirantis (12,6 %), Rackspace (7,4 %), IBM (4,9 %) a HP (4,7 %).

[13] Ansible je open-source automatizační nástroj od firmy RedHat, který pomocí playbooků provádí nejrůznější úkoly jako např. konfigurace systémů, nasazování softwarů, orchestrace složitých IT úkolů a mnohem více... [23].

1.5.1 Architektura

V Openstacku mají uzly až 5 rolí[14]: řídící, výpočetní, síťová, úložný a monitorující, přitom první 3 jsou pro nasazení cloudu povinné. Tyto role pak nabírají různé funkce v závislosti na vybraných službách OpenStacku (taktéž lze označovat jako komponenty OpenStacku). Architekturu pak tvoří vybrané služby a přiřazení rolí daným uzlům. Obecně může architektura vypadat jako na obrázku 1.12 (Pro větší přiblížení skutečnosti existuje logická architektura v online dokumentaci OpenStacku[15]).



Obr. 1.12: Ukázka zjednodušené architektury OpenStacku

Na **kontroléru** zpravidla běží služby: identity, diskových obrazů, rozmístění, orchestrace, řízení částí výpočetních uzlů, správy sítí (včetně firewallu a bezpečnostních skupin, viz kap. 1.5.4), různých síťových agentů a webové GUI (Dashboad). Taktéž zahrnuje SQL databázi, frontu zpráv[16] a NTP[17] (angl. Network Time Protocol). Volitelně může Kontrolér obsahovat i služby pro blokové a objektové uložiště

[14] V závislosti na použitém instalačním nástroji může existovat šestá role – Nasazující (angl. Deployments). Na uzlu je pouze instalační nástroj, který nasazuje OpenStack na všechny ostatní uzly. Provádí se z něj aktualizace, přidávání dalších uzlů, zálohy, ... Ve výchozím stavu se tato role přiřazuje kontroléru.

[15] `https://docs.openstack.org/install-guide/get-started-logical-architecture.html`

[16] Předává požadavky od klientů příslušným výpočetním uzlům a po dokončení jim vrací výsledky zpět – RabbitMQ.

[17] Všechny uzly v cloudu musí mít pro správnou komunikaci synchronizovaný systémový čas.

a monitorování. V rámci rozsáhlých cloudů může OpenStack využívat další kontroléry pro vyvažování zátěže, či zaručení vysoké dostupnosti. Role monitorování a síťování je taktéž možné nasadit na další uzly [27].

Výpočetní uzly poskytují své HW prostředky kontroléru a na základě příchozích požadavků vytvářejí VMs nebo kontejnery. Tyto instance jsou vytvářeny implementovaným hypervisorem, nebo běhovým prostředím – implicitně využívá KVM-QEMU a Docker. Dále na těchto uzlech běží služba síťového agenta, který připojuje instance do virtuálních sítí (samotné virtuální sítě jsou vytvářeny uzlem se síťovou rolí).

Volitelně může OpenStack využívat dvou **uložišť** [27]:

- **Blokové** – obsahuje disky, které blokové uložiště a služba sdíleného souborového systému (Cinder) poskytuje pro instance. V nasazení může být použito více než jeden uzel.
- **Objektové** – obsahuje disky, které služba objektového uložiště (Swift) využívá pro ukládání účtů, kontejnerů a objektů. Vyžaduje v nasazení alespoň 2 uzly.

Datový přenos služeb mezi výpočetními a těmito uzly využívá managementovou síť. Role uložiště může být taktéž předána kontroléru (například použitím LVM, nebo připojením externího disku) [27].

Všechny uzly vyžadují přístup k internetu pro administrativní účely jako jsou instalace různých balíčků, bezpečnostní aktualizace, DNS a NTP. OpenStack vyžaduje minimálně dvě fyzické sítě:

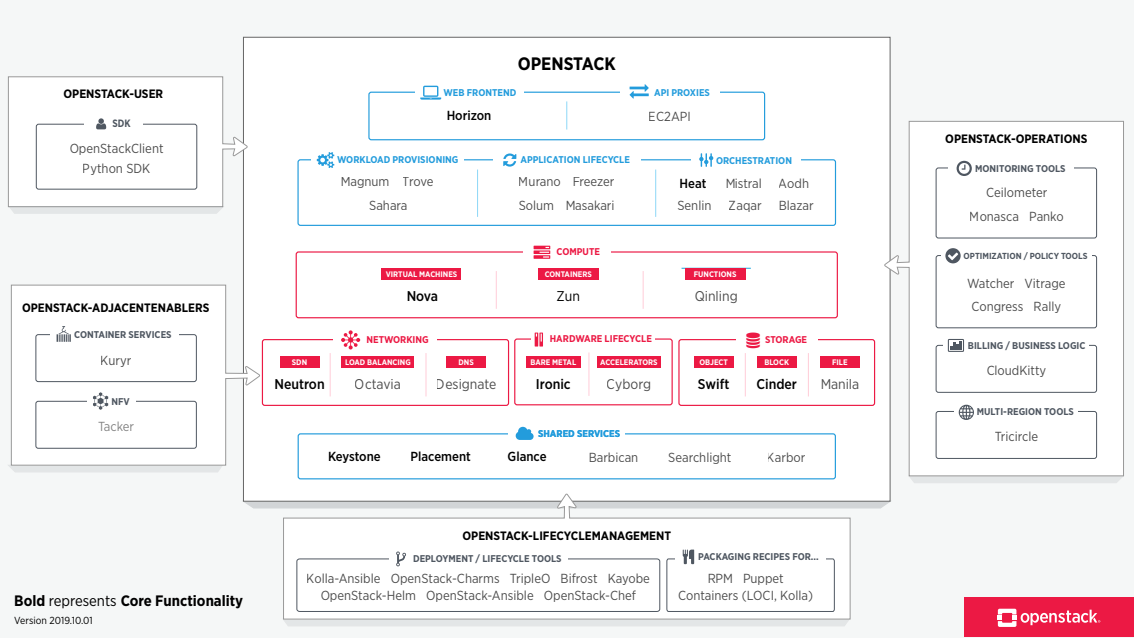
- **Síť managementu** (angl. Management Network, pro všechny uzly) – zprostředkovává konektivitu mezi všemi uzly pro účely instalace, konfigurace, nebo komunikace vnitřních procesů OpenStacku.
- **Síť poskytovatele** (angl. Provider Network, pouze pro síťové a výpočetní uzly) – Poskytuje připojení k internetu pro vytvořené instance.

Provoz sítí managementu, nebo poskytovatele je taktéž možné posílat do internetu přes NAT, avšak například by byl znemožněn přístup z veřejné sítě na vytvořené instance (teoreticky by přístup byl možný skrz složitější manuální nastavování DNAT a SNAT). Obecně se doporučuje natovat síť managementu (z důvodu bezpečnosti) a pro síť poskytovatele používat adresy z rozsahu veřejných IP.

1.5.2 Služby OpenStacku

Funkčnost a možnosti OpenStacku závisí hlavně na sestavené architektuře dle vybraných služeb (z celkem 43 možných). Dostupné služby, nástroje a rozhraní zobrazuje ve funkčním rozdělení obrázek 1.13.

Základní povinné služby OpenStacku jsou: Keystone, Nova, Glance a Neutron, které samy o sobě umožní především: vytvářet VMs, směrovače a virtualizované



Obr. 1.13: Mapa OpenStacku (převzato z [24])

sítě, spravovat diskové obrazy, bezpečnostní skupiny, migrace VMs v reálném čase a rozdělovat tyto celé struktury do projektů spolu s vytvářením uživatelů – vše skrz příkazový řádek OpenStack klienta. Od verze Stein je povinná navíc i služba Placement [27].

Stručný popis hlavních služeb popisuje tabulka 1.1. Avšak pro splnění všech potřeb pouze základní služby většinou nestačí – výběr dalších velmi užitečných služeb popisuje tabulka 1.2.

Tab. 1.1: Stručný přehled hlavních služeb OpenStacku [24, 25]

Služba	**Popis**
Keystone	Služba identity, poskytuje autorizační služby a řízení přístupu (RBAC) pro OpenStack komponenty.
Glance	Spravuje diskové obrazy pro instance, vytváření snapshotů.
Nova	Výpočetní služba k vytváření a správě VMs.
Neutron	Síťová služba pro vytváření sítí, směrovačů, firewallů a mnoho dalšího, včetně IP adresace, směrování a SDN.
Placement	Sleduje umístění a využití cloudových zdrojů, data pak předává dle potřeb dalším službám (např. alokace množin IP, využití výpočetních zdrojů).

Tab. 1.2: Stručný přehled volitelných služeb OpenStacku [24, 28]

Služba	**Popis**
Horizon	Poskytuje webové GUI pro práci s OpenStackem.
Cinder	Blokové uložiště pro vytváření volumů.
Swift	Objektové uložiště.
Heat	Poskytuje orchestraci cloudových aplikací založených na šablonách pro prostředí OpenStacku.
Zun	Kontejnerová služba s podporou různých běhových prostředí.
Kuryr	Docker síťový plugin, který spojuje vytvořené kontejnery se síťovou infrastrukturou Neutronu.
Magnum	Využívá Heat k orchestraci kontejnerových infrastruktur pomocí COEs (Container Orchestration Engines).
Ceilometer	Poskytuje sadu nástrojů pro měření a monitorování komponent OpenStacku.
Rally	Testovací nástroj, který automatizuje testování a ověřování nejrůznějších scénářů, na základě výsledků mění konfigurace vedoucí ke zlepšení SLA, výkonu a stability.
Murano	Poskytuje katalog cloudových aplikací, které nasazuje pomocí orchestračních šablon. Automaticky vytváří VMs s běžícím Dockerem nebo k8s, aplikace pak nasazuje do jejich kontejnerů nebo podů. Data aplikace zabezpečuje službou Barbican.
Barbican	Poskytuje zabezpečené uložiště (např. pro hesla, šifrovací klíče).

1.5.3 Kontejnerizace

Možnosti kontejnerizace byly přidány do OpenStacku od verze Mitaka a nabízí celkem 3 řešení (2 v současnosti). První možnost nabízela služba Nova doinstalováním ovladače nova-docker s následnou změnou v konfiguraci tak, aby využívala pro nasazování serverů Docker, místo výchozího: libvirt s KVM. Pro ukládání Docker obrazů bylo nutné přenastavit i Glance [29]. Nicméně toto řešení omezovalo vytváření VMs, tak bylo přemístěno jen do instalace DevStack, ale i tak jeho podpora skončila ve verzi Ocata (více na stránkách opendev.org[18]).

Zun

Od verze Queen byl Nova-Docker úplně nahrazen novým projektem – Zun, který poskytuje API pro nasazení aplikací do kontejnerů bez potřeb vytváření a správy serverů, nebo clusterů. Ke svému fungování požaduje minimálně služby [30]:

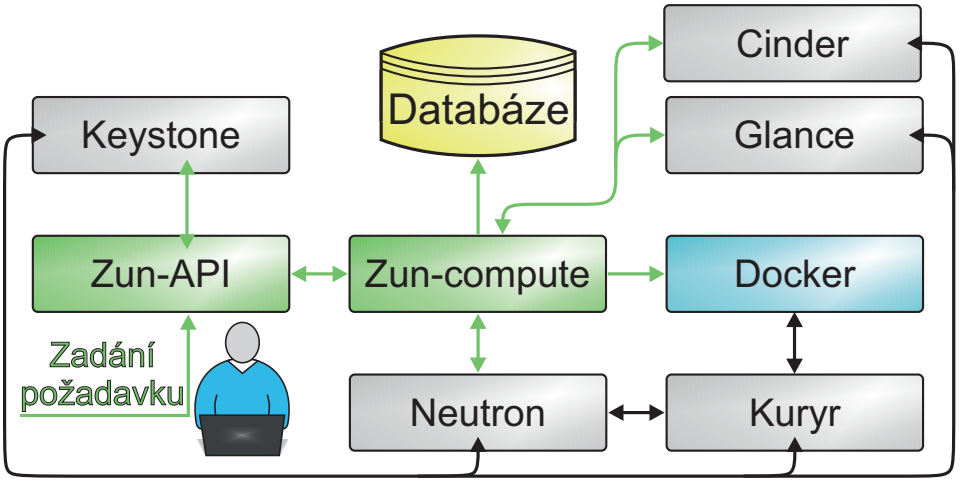
[18] `https://opendev.org/x/nova-docker`

- **Keystone** – pro ověřování požadavků a vyhledávání dalších služeb.
- **Neutron** – pro DHCP a konfiguraci sítě.
- **Kuryr** – propojuje vytvořené kontejnery s infrastrukturou Neutronu.
- **Placement** – sledování zdrojů a umístění kontejnerů.

Dále jej podporují i služby:

- **Cinder** – volumy pro kontejnery.
- **Glance** – ukládání obrazů kontejnerů.
- **Horizon** – GUI pro Zun.
- **Heat** – orchestrace kontejnerů.

Obrázek 1.14 zobrazuje příkladné umístění Zunu v architektuře OpenStacku a jeho komunikační procesy. Celý řetězec akcí začíná zasláním požadavku `zun-api` na vytvoření kontejneru přes CLI nebo Horizon s autentizačním tokenem (těsně předtím se musí uživatel autorizovat u Keystonu, který mu vygeneruje dočasný token). `Zun-api` po obdržení požadavku zašle přijatý token Keystonu k validaci, který obratem zašle zpět autentizační hlavičky s rolemi a oprávněními daného uživatele. `Zun-api` zkontroluje zda má uživatel pravomoc vykonat jeho požadavek. Pokud vše proběhlo v pořádku, `zun-api` kontroluje další řadu kroků: jestli existuje cílová síť, volný port, obraz v lokálním uložišti, ... zkrátka shromažďuje vše potřebné na základě požadavku. Nakonec předá výsledek API správci výpočetních uzlů a ten vybere na základě plánovače (angl. Scheduler) cílový uzel pro nasazení nového kontejneru. `Zun-compute` si vyzvedne žádost z fronty zpráv (AMQP – Advanced Message Queuing Protocol) a pokračuje ve vytvoření kontejneru: stáhne obraz z Glance (nebo později předá v požadavku pro Docker, aby si jej stáhl z Docker Hubu), vyčlení prostředky požadované pro kontejner, aktualizuje si svoji tabulku dostupných zdrojů,



Obr. 1.14: Zun v architektuře OpenStacku

atd. Následně je konečně požadavek zaslán Dockeru. Pokud je kontejner úspěšně vytvořen, `zun-compute` změní status kontejneru na „created“ a uloží objekt kontejneru do `databáze`. Objekt kontejneru uchovává informace jako použitý diskový obraz, příkaz, souborový systém, zvolený výpočetní uzel a podobně [28].

Podobně jako pody v k8s, Zun umožňuje vytvářet tzv. kapsle. Tato jednotka obsahuje jeden sandbox kontejner[19], jeden, nebo více kontejnerů a jeden, nebo více volumů. Všechny kontejnery uvnitř kapsle jsou nasazeny na stejném uzlu a sdílejí zdroje jako síť, IP adresu, jmenný prostor a cgroups [28, 30].

Magnum

Magnum se stal prvním samostatným projektem, který umožňoval vytvářet kontejnery nad vytvořenou infrastrukturou. Aktuálně podporuje COEs: Kubernetes, Apache Mesos a Docker Swarm. Funkčnost Magnumu je závislá na dalších službách: Keystone, Heat, Nova, Neutron a Glance. Například Neutron poskytuje propojení clusteru s infrastrukturou OpenStacku a uvnitř clusteru zase řeší síťování jeden z ovladačů: Flannel, Docker, nebo Calico [28, 31].

Před vytvořením clusteru je nutné specifikovat jeho šablonu. Šablona obsahuje informace o clusteru jako jsou diskové obrazy, šifrovací klíče, prostředí, počet CPU jader a velikost RAM pro kontroléry a výpočetní uzly, velikost volumu, atd. Následně lze při vytváření clusteru specifikovat počet kontrolérů a výpočetních uzlů. Požadavek je dále zpracováván službou Heat, která na základě šablony vytvoří cílový cluster s veškerou infrastrukturou na VMs nebo bare metal servery. Uživatelé pak ovládají cluster připojením se na VM kontroléru a používáním CLI již dle zvoleného COE. Další z hlavních vlastností projektu Magnum jsou [28, 31]:

- má schopnost zvyšovat nebo snižovat počty uzlů v clusteru, bez ovlivnění funkčnosti nasazených kontejnerů,
- podporuje více projektů pro kontejnery clusteru,
- může využít Cinder volumy pro kontejnery,
- bezpečný přístup do kontejnerů v clusteru pomocí TLS,
- podporuje externí infrastrukturu, která může být využívána clusterem jako DNS, veřejná síť, Docker registr, vyvažování zátěže atd.,
- Barbican poskytuje uložiště pro důvěrné informace jako jsou certifikáty TLS v rámci clusteru,
- spolupráce s Kuryrem,
- monitorování kontejnerů (spotřeba CPU v rámci celého uzlu nebo kontejneru, paměti, atd.).

[19]Sandbox kontejner je vytvořený pouze jako síťový uzel pro ostatní kontejnery uvnitř kapsle.

Sandboxing

Pro podporu sandboxingu spolupracuje nadace OpenStacku (angl. OpenStack Foundation – OSF) s komunitním projektem Kata Containers[20]. Kata kontejnery mají podporu v projektech Zun, Magnum a i v externích platformách jako jsou k8s nebo OpenShift. Přichází s novým běhovým prostředím `kata-runtime`. Do verze Train se toto nové běhové prostředí manuálně doinstalovávalo na výpočetní uzly. V nové verzi Ussuri by měla být možnost Kata kontejnery implementovat již při nasazení OpenStacku [32].

Technologie Kata kontejnerů kombinuje největší výhody z kontejnerů (rychlost) a VMs (bezpečnost). Běhové prostředí Kata prvně vytvoří obálku – velmi odlehčenou verzi VM, který je vytvořený hypervisorem. Následně do obálky vytvoří 1 nebo více kontejnerů (záleží od technologie) [32].

Teoreticky se dá předpokládat, že Magnum již v základu poskytuje částečné sandboxové prostředí. Magnum totiž vytváří cluster v podobě virtuálních strojů a teprve až v nich se vytváří kontejnery/pody. Oddělení podů od operačního jádra hostitele tedy plně splňuje. Otázkou zůstává, do jaké míry se pody dokážou ovlivňovat mezi sebou.

1.5.4 Síťování

Hlavní role v síťování představuje služba Neutron. Pro síťování kontejnerů je pak potřeba i služby Kuryr, která propojuje Docker s infrastrukturou Neutronu.

Neutron

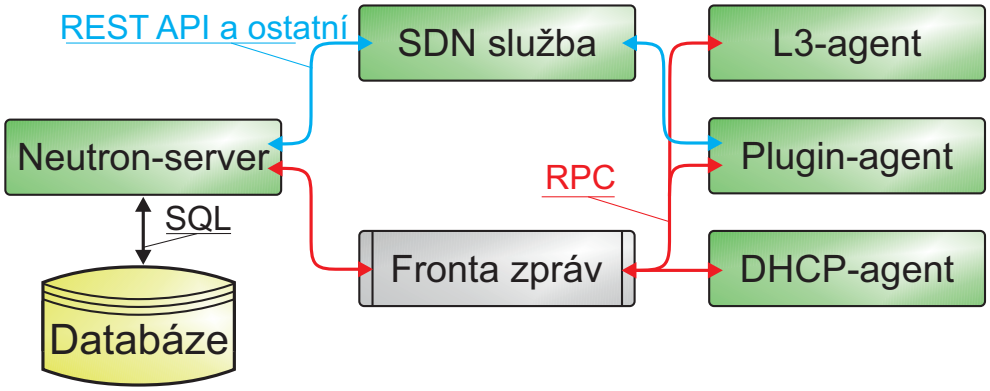
Služba Neutron poskytuje „network connectivity as a service“ mezi virtuálními rozhraními (vNICs – virtual Network Interface Cards) spravovanými dalšími službami OpenStacku, jako například Nova a Zun. Neutron obsahuje v základu tyto komponenty [33]:

- **Neutron server** (`neutron-server` a `neutron-*-plugin`) – zpracovává příchozí požadavky na síťové API a jeho rozšíření. Taktéž vytváří model sítě a IP adresaci pro každý port. Komunikuje s SQL databází použitím AMQP.
- **ML2 plugin agent** (`neutron-*-agent`) – běží na každém výpočetním uzlu k správě konfigurace lokálního virtuálního přepínače (vswitch). Instalace to-

[20]Pozn.: Kata kontejnery vznikly z předchůdce *Clear Containers*, tento projekt byl kompletně zmigrován do Kata kontejnerů a přestal být nadále vyvíjen, viz informace v repositáři `clearcontainers/runtime` na stránkách GitHubu: `https://github.com/clearcontainers/runtime`.

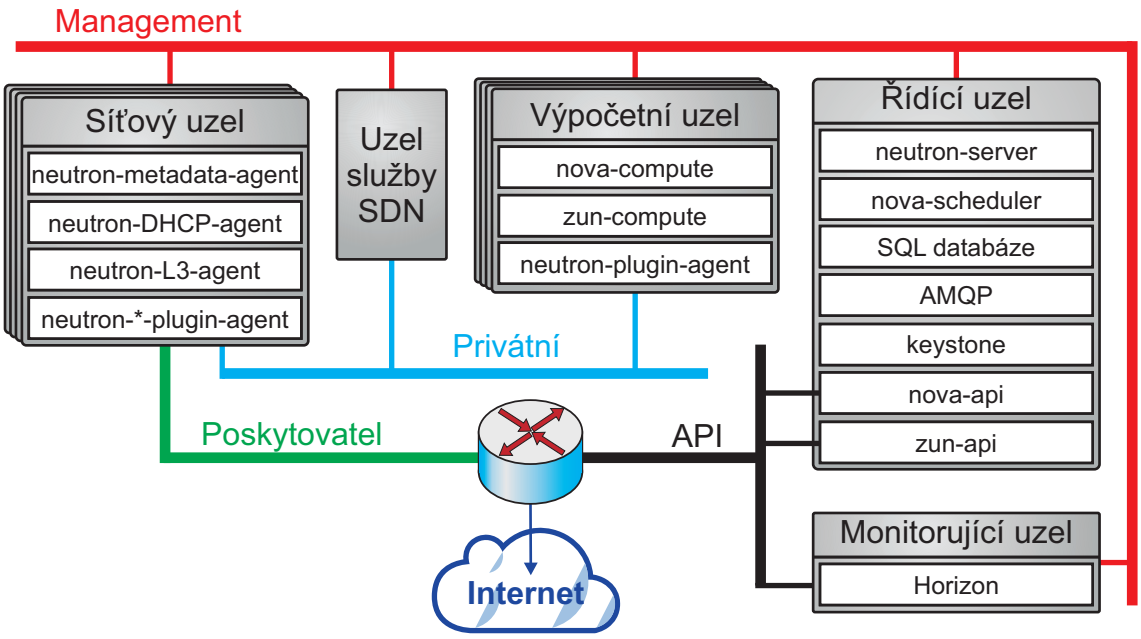
hoto agenta závisí na zvoleném pluginu (dle toho implementuje i L2 population[21]):
  - openvswitch (OVS),
  - linuxbridge,
  - a dále: vmware_nsxv, vmware_nsxv3, vmware_dvs, opendaylight.
- **DHCP agent** (`neutron-dhcp-agent`) – poskytuje DHCP službu pro všechny tenant sítě.
- **L3 agent** (`neutron-l3-agent`) – poskytuje zasílání L3/NAT provozu instancí v tenant sítích pro přístup do externí sítě.
- **Služby sítě poskytovatele** (*SDN server/services*) – poskytuje pokročilejší síťové služby pro tenant sítě.
- **Fronta zpráv** – směruje zprávy mezi procesy Neutronu napříč uzly (pom. AMQP).
- **Databáze** – SQL databáze pro ukládání dat.

Architekturu Neutronu reprezentuje obrázek 1.15. Následné umístění komponent Neutronu na fyzických uzlech zobrazuje obrázek 1.16 spolu s pár dalšími službami v OpenStacku (Zun, Nova, Keystone). Privátní síť přestavuje vytvořenou virtuální infrastrukturu sítí. API síť odhaluje všechny API OpenStacku pro tenanty a může být sloučena i se sítí poskytovatele. `Zun-compute` a `nova-compute` jsou zodpovědní za vytváření instancí na výpočetních uzlech. Z obrázku vyplývá, že pokud budou chtít tyto instance komunikovat do internetu, provoz musí vždy procházet přes síťový uzel. [33].



Obr. 1.15: Architektura Neutronu

[21] L2 population je speciální mechanizmus, který optimalizuje BUM (Broadcast, Unkown destination address, Multicast) provoz v překryvných sítích VXLAN a GRE. Potřebný hlavně ve spojení s pluginy linuxbridge nebo openvswitch.



Obr. 1.16: Využívání komponent Neutronu z pohledu fyzických uzlů

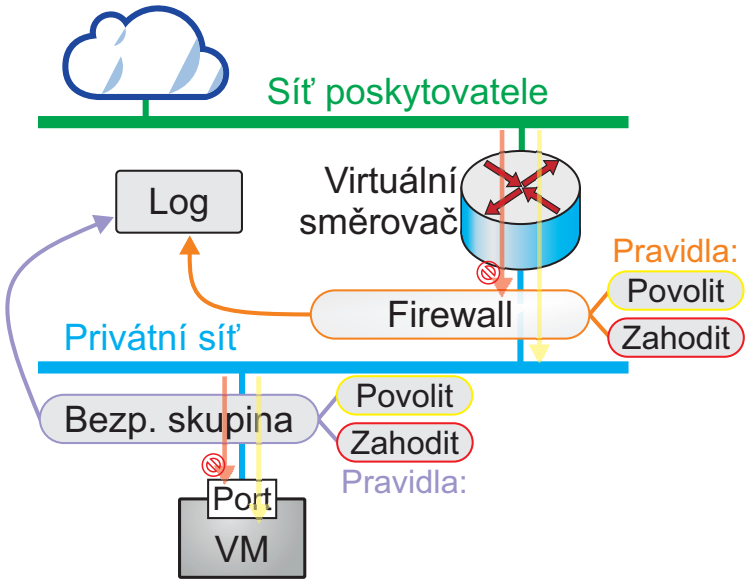
Neutron poskytuje v základní instalaci: vytváření sítí (v závislosti na zvoleném ML2 pluginu: lokální, flat, vxlan, GRE), směrovačů, adresaci v rámci IPv4/IPv6, NAT (DNAT, SNAT, plovoucí IP adresy[22]), nastavování bezpečnostních skupin, DNS, DHCP, atd. V závislosti na přidaných modulech může Neutron využívat i pokročilejších funkcí [34]:

- `VPNaaS` – Virtual Private Network as a Service,
- `SR-IOV` – Single Root I/O Virtualization,
- `DVR` – Distributed Virtual Router,
- `port-forwarding` – Nástavba pro plovoucí IP umožňující navíc i přesměrovávat volný port ze síťového uzlu na konkrétní port aplikace běžící v instanci,
- `FWaaS` – Firewall as a Service,
- `QoS` – Quality of Services,
- `agent-HA` (High Availability) – vysoká dostupnost pro DHCP,
- `BGP-DRagent` – Border Gateway Protocol Dynamic Routing agent,
- `provider-networks` – poskytuje konektivitu na úrovni L2 mezi instancemi a fyzickou sítí použitím VLAN,
- `segments` – rozšíření umožňující operace CRUD (Create, Read, Update, Delete) pro modul `provider-networks`,

[22] Angl. Floating IP je uvolněná adresa ze sítě poskytovatele, která může být přiřazená zvolenému portu, např. vytvořené instanci v privátní síti a tím ji tak udělá dostupnou z venkovní sítě. Využívá techniku NAT one-to-one [34].

- `SFC`[23] – podporuje Service Function Chaining v Neutronu,
- `metering` – Umožňuje IP rozsahům přiřazovat popisky (angl. Labels) a dle nich shromažďovat data ohledně využití přenosového pásma, které zasílá na oznamovací systém Oslo. Může být využit službou Ceilometer.

V rámci nainstalovaných pluginů může Neutron splňovat obrovské množství scénářů. Přidáním pluginu FWaaS získá schopnost vytvářet pravidla do bezpečnostních politik, které se následně přidávají do firewallů. Vytvořené firewally se mohou aplikovat na jakékoliv porty v rámci vnitřní infrastruktury OpenStacku. Rozdíl mezi firewally a bezpečnostními skupinami je patrný z obrázku 1.17. Bezpečnostní skupiny se používají pouze na koncových portech instancí a zároveň jsou vždy povinné. Firewally disponují větší dynamičností z hlediska jak ve vytváření pravidel, tak i místě použití. [34].



Obr. 1.17: Uplatnění firewallu a bezpečnostních skupin

Kuryr

Kuryr propojuje Docker libnetwork s API Neutronu a poskytuje schopnost připojit VMs, kontejnery a bare metal servery do stejné virtuální sítě. Použití Neutronu jako poskytovatele pak umožňuje kontejnerům využívat funkce jako: Bezpečnostní skupiny, množiny podsítí, NAT, port security (ARP spoofing), QoS, správu kvót, IPAM, FWaaS [28]. Pro propojení volumů (např. ze služby Cinder) se stará projekt

[23] SFC je mechanismem „přepisování odesílání paketů“ na základě cílové destinace, ekvivalentem k technice Police Based Routing ve fyzických IP sítích.

Fuxi, který se v roce 2016 stal subprojektem Kuryru [35] – není potřeba jej dodatečně instalovat.

1.5.5 Minimální HW požadavky

Pro nasazení plně funkčního OpenStacku je potřeba minimálně dvou uzlů (kontrolér a výpočetní uzel). Pokud budeme uvažovat rozpoložení architektury jako na obrázku 1.12, uzly by měly v produkčním nasazení splňovat minimálně požadavky uvedené v tabulce 1.3 [27].

Tab. 1.3: Minimální HW požadavky uzlů OpenStacku [27]

Uzly	**CPUs**	**RAM**	**Disk**	**NICs**
Kontrolér	1–2	8 GB	100 GB	2
Výpočetní uzel	2–4+	8+ GB	100+ GB	2
Blokové uložiště	1–2	4 GB	100+ GB	1
Objektové uložiště	1–2	4+ GB	100+ GB	1

1.5.6 Instalace

Manuální instalace OpenStacku je značně komplikovaná (avšak ne nemožná) a velmi zvyšuje riziko špatné konfigurace některých komponent způsobené chybou lidského faktoru. Proto již existuje celá řada nástrojů[24], které nasazování na servery značně ulehčují. V základu se dá nasazení OpenStacku rozdělit do dvou možností:

1. all-in-one,
2. multinode.

Metoda *all-in-one* spočívá v nasazení všech služeb OpenStacku na jediný uzel. Obecně se ale tato metoda nedoporučuje do produkčního nasazení, nýbrž pouze pro testovací účely. Samotné služby OpenStacku mohou být navíc nasazeny do kontejnerů, což přináší další výhody vyplývající z kontejnerizace jako takové. Služby OpenStacku lze taktéž nainstalovat přímo na uzly pomocí balíčkovacího systému (angl. Package Manager) – např. `rpm`, `apt`.

Instalační nástroj **DevStack** je navržen přesně pro účely nekontejnerizovaného *all-in-one* nasazení. Využívá různých skriptů k jednoduchému a rychlému nasazení

[24]Všechny nástroje jsou dostupné na stránkách openstack.org: `https://www.openstack.org/software/project-navigator/deployment-tools`.

a nastavení prostředí OpenStacku. Instalace je založená na verzi `master`[25] bez potřeby pokročilých znalostí OpenStacku. Minimální HW požadavky pro DevStack jsou [36]:

- 6+ GB RAM,
- 30+ GB volného místa na disku.

Mnohem propracovanějším nástrojem je **Kolla-ansible**. Podporuje *all-in-one* i *multinode* možnosti nasazení do Docker kontejnerů. Všechny obrazy kontejnerů jsou spravovány dalším nástrojem se zkráceným názvem **Kolla** – implementuje příkaz `kolla-build`, který přímo vytváří obrazy kontejnerů pro většinu projektů OpenStacku na distribucích Linuxu: CentOS, Ubuntu, Debian a Rhel. Vytváření těchto obrazů může být typu `source` (instalace bude provedena ze zdrojového kódu), nebo `binary` (instalace pomocí balíčkovacího systému). Hostující uzel (uzel s rolí *nasazující*) by měl disponovat minimálně [28, 37]:

- 8 GB RAM,
- 40 GB volného místa na disku,
- dvěma síťovými rozhraními.

Po nainstalování všech závislostí se nasazení OpenStacku odvíjí od konfigurace dvou souborů [37]:

- **soubor ansible inventáře** (angl. Inventory File) – specifikuje *„jaká role uzlu má být nasazena na jaký server“*. Každé roli jsou již ve výchozím stavu přiděleny patřičné služby OpenStacku.
- **globals.yml** – hlavní konfigurační soubor určující vlastnosti OpenStacku. Lze volit jaké služby má obsahovat, verzi vydání, rozhraní pro sítě managementu a poskytovatele, atd.

Kolla-ansible pak dle nastavení spouští úkoly ansible playbooků. Pro nasazení je použito tří playbooků. První přednastavuje všechny cílové uzly, instaluje potřebné balíčky/aplikace, ... Druhý kontroluje správnou konfiguraci souborů a zda prostředí systému splňuje všechny požadavky pro nastavené volby v **globals.yml**. Pokud playbook projde bez chyby, lze pokračovat k třetímu, který nasazuje a instaluje samotné služby OpenStacku včetně s jejími základními konfiguracemi [37].

1.6 OpenShift

Samotný k8s poskytuje: možnosti kontejnerové orchestrace, odolnost podů proti selhání a nasazení služeb a aplikací. Ke správnému fungování ale potřebuje spoustu

[25] Master je poslední verzí v git repositáři. V OpenStacku toto vydání představuje jeho následující verzi (Victoria), která je ovšem ve fázi vývoje. Tudíž může dojít k nesprávné funkčnosti některých komponent.

dalších komponent (např. implicitně neposkytuje SDN, nebo metodu k řízení síťového provozu komunikace s/mezi pody). Řešení pak závisí na konkrétním administrátorovi, aby do k8s přidal další nástroje a naučil se s nimi navíc i zacházet. OpenShift je distribucí Kubernetes, která všechny potřebné rozšiřující vlastnosti (projekty komunity) sjednocuje. V současné době kombinuje OpenShift více než 200 open-source projektů [13].

OpenShift vyvinula společnost RedHat roku 2011, která aktuálně nabízí 4 placené produkty [38]:

- **OpenShift on IBM Cloud** (dříve známý jako OpenShift Online) – hostovaný na serverech RedHatu a IBM.
- **OpenShift Dedicated** – hostovaný na serverech AWS (angl. Amazon Web Services), nebo cloudu Google.
- **OpenShift Container Platform** (dříve známý jako OpenShift Enterprise) – hostovaný na vlastních serverech.
- **Azure Red Hat OpenShift** – hostovaný na serverech Microsoft Azure.

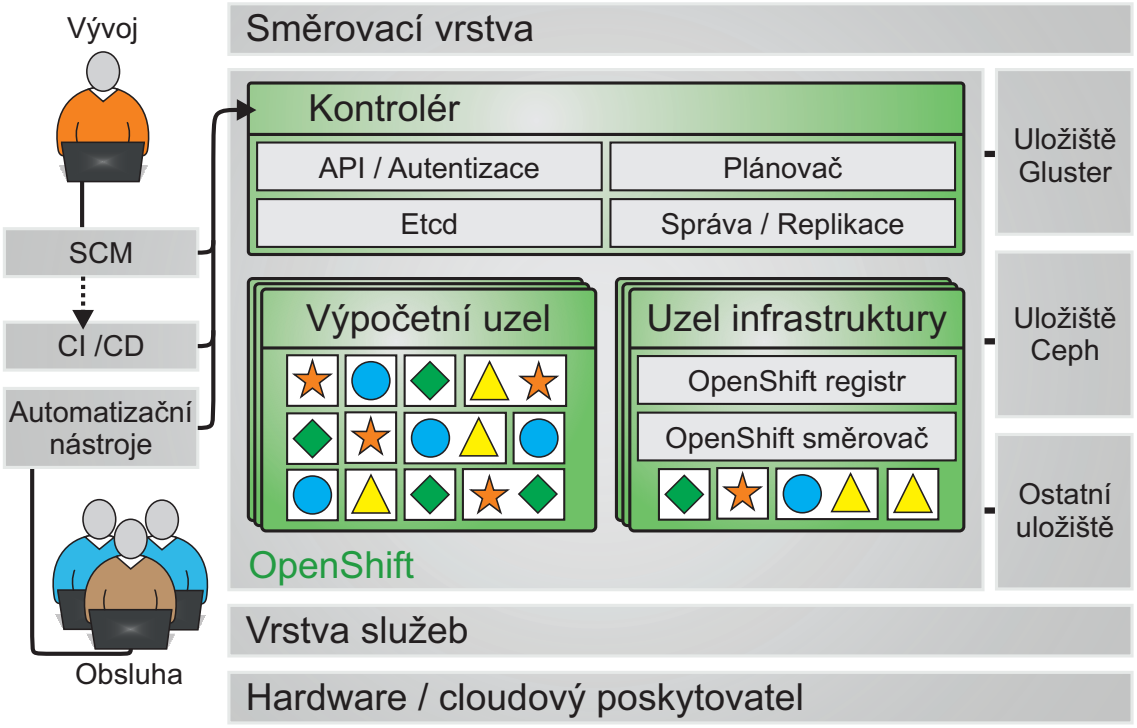
a jeden open-source projekt – **OKD** (angl. Origin Community Distribution), dříve známý pod názvem OpenShift Origin. Dle [39] se na vývoji OKD (včetně dokumentace a vedlejších modulů) podílí nejvíce společnost RedHat (83,7 %) a nezávislí uživatelé (15 %). Vlastnosti projektu OKD jsou převzaty do placených verzí zmíněných výše, které jsou přizpůsobeny do produkčního nasazení. Všechny verze OpenShiftu mají shodné vlastnosti, liší se hlavně v bohatosti poskytované podpory (basic, standard, premium), možnostech využít vlastní nebo externí HW zdroje, počtu oddělených projektů a možnostech přizpůsobit nasazení dle požadavků zákazníka. OKD je samozřejmě bez podpory a vlastník serverů je odkázán na administrátora se znalostmi použitých komunitních projektů. Placené verze pak garantují: uživatelskou podporu včetně problémů s Dockerem a k8s, bezpečnost a stabilitu systému [38, 40].

1.6.1 Architektura

OpenShift je postaven nad k8s a oba projekty tedy mají většinu komponent architektury společné (architektura k8s je k dispozici v kap. 1.4.1), jako jsou [13]:

- **OpenShift kontrolér** – podobný jako Master v k8s, avšak poskytuje spoustu funkcí navíc, viz kap. 1.6.2.
- **OpenShift výpočetní uzly** – stejné jako uzly Workers v k8s ke spouštění kontejnerizovaných aplikací. OpenShift navíc obsahuje služby pro poskytování síťové konektivity mezi pody, DNS, monitorování uzlu a agregaci logů.
- **Etcd uložiště**.

OpenShift přichází navíc s tzv. **uzly infrastruktury**, které obsahují stejné komponenty jako každý k8s Worker a navíc OpenShift směrovač, Openshift interní registr a popřípadě monitorující řešení, které sleduje stav clusteru a všech služeb, které na něm běží. Stručnou architekturu zobrazuje následující obrázek 1.18.



Obr. 1.18: Pohled na OpenShift architekturu

1.6.2 Vlastnosti platformy OpenShift

OpenShift využívá vlastností k8s, sjednocuje je dohromady a speciálně se tak zaměřuje na usnadnění použití pro vývojáře i vlastníky aplikací. OpenShift může být taktéž nasazen v kombinaci s OpenStackem a využívat tak nejlepších vlastností z obou platforem. Oproti k8s přináší OpenShift, v závislosti na povolených službách, následující funkce [13, 40] (řeší: K – kontrolér; VUI – uzel infrastruktury; ALL – všechny uzly):

- **Autentizace** (K) – použití autentizace je vyžadováno pro uživatele, provádění příkazů, projekty, uzly, atd. OpenShift poskytuje metody: lokální, LDAP[26], na základě HTTP hlavičky požadavku, Keystone (pokud je instalován s OpenStackem), GitHub jako poskytovatel identity, generování tokenů a další [41].

[26]Lightweight Directory Access Protocol – ukládání a přístup k datům organizovány ve stromové struktuře.

- **Multi-Tenancy** (K) – umožňuje vytvářet projekty a uživatele, rozdělit různé scénáře do projektů a k nim přiřazovat přístupy různým uživatelům.
- **Webové GUI** (K).
- **Image Builder** (ALL) – proces, který je použit k transformaci parametrů kontejnerového obrazu, nebo zdrojových kódů Gitu, do spustitelného obrazu.
- **CI**[27]**/CD**[28] (-) – OpenShift poskytuje flexibilní integraci se softwarem Jenkins.
- **SCM** (angl. Source Control Management) (-) – Nástroj pro vývojáře, umožňující verzovat zdrojový kód. OpenShift v tomto případě využívá SCM ve spojení s GitHubem.
- **REST API** (ALL) – oproti k8s, OpenShift poskytuje své vlastní API, které využívá všech funkcí automatizace a integrace s externími platformami.
- **Interní registr** pro obrazy kontejnerů (VUI).
- **OpenShift směrovač** (VUI) – Odhaluje IP adresy a porty aplikací pro externí uživatele (klienty). Vyvažuje zátěž mezi pody aplikace v závislosti na příchozím provozu.
- Již implementovaný **CoreDNS** (K).
- **OpenShift SDN** (ALL) – řeší síťování v clusteru dle povoleného SDN pluginu.
- **Prometheus** a **Grafana** (VUI)– komponenty OpenShiftu od verze 3.11 pro účely monitorování. Prometheus shromažďuje informace o stavu clusteru a běžících komponentách. Grafana vykresluje získané data do webového GUI.
- **Elasticsearch** a **Kibana** (VUI) – Elasticsearch shromažďuje všechny logy CNI pluginu a Kibana opět prezentuje agregovaná data ve webovém GUI.

K uzlům kontroléru přistupují vývojáři, administrátoři a vlastníci aplikací pomocí CLI, nebo GUI navíc s možností využít automatizačních, nebo CI/CD nástrojů [13].

1.6.3 Kontejnerizace

OpenShift využívá k nasazení aplikace do kontejnerů platformu Kubernetes (kapitola 1.4.2), avšak z pohledu vývojářů nabízí zcela nový postup. Při vytváření obrazu z vlastního kódu, převezme zdrojový kód a další práci CI server (v tomto případě obvykle Jenkins). Ten následně provede vytvoření obrazu, pošle jej do registru a zároveň vytvoří na výpočetních uzlech pody dle popisu aplikace. Při změně zdrojového kódu CI server aktualizuje předešlý obraz, opět zašle novou verzi do registru a následně na výpočetních uzlech smaže staré pody a vymění je za nové – již s upraveným obrazem [13, 41].

[27] Angl. Continuous Integration – je technika při vytváření softwaru. Změna v kódu provedená vývojářem nebo dalšími členy v týmu je několikrát denně integrována do společného repositáře, kde je obvykle zkontrolována automatizační testovací sadou [42].

[28] Angl. Continuous Delivery – je technika úzce spjatá s CI, která zaručuje spolu s Continuous Development velmi rychlé nasazení zvolené verze (pokud prošla automatizačními testy) [43].

Openshift používá nejčastěji kontejnerové běhové prostředí **CRI-O**, které poskytuje kompatibilitu pro spuštění běhových prostředí jako runc, implicitní OCI-runtime, nebo Kata kontejnery [41].

Sandboxing

Pro podporu sandboxingu vychází OpenShift z technologií dostupných pro k8s, viz sekce **Sandboxing** v Kubernetes.

1.6.4 Síťování

Z hlediska dosažitelnosti služeb může být použit DNS server, který implicitně naslouchá na kontroléru. Příchozí pakety směruje směrovač OpenShiftu na příslušné `kube-proxy` [13].

OpenShift využívá k zajištění jednotné sítě clusteru technologii SDN, která nastavuje možnosti komunikace mezi pody pomocí překryvných sítí VXLAN dle zapnutého SDN modulu [13, 40, 41]:

- **OpenShift ovs-subnet** – stejné jako v k8s, všechny pody spolu mohou komunikovat.
- **OpenShift ovs-multitenant** – ve výchozím stavu spolu mohou komunikovat pouze pody v rámci stejného projektu (každý projekt má přidělený unikátní VNID – Virtual Network ID). Pro povolení komunikace mezi projekty slouží příkaz `join-projects`, nebo lze všem projektům jednosměrně povolit příchozí komunikaci do podů daného projektu příkazem `make-projects-global` (všechny globální projekty mají VNID 0, jako například projekt „*default*“).
- **OpenShift ovs-networkpolicy** – implementuje schopnosti k8s síťové politiky. Ve výchozím stavu se chová jako *ovs-subnet*, zato při nastavení politiky `deny-all` se chová jako *ovs-multitenant*. Tento modul umožňuje správcům nastavit vlastní, mnohem přesnější pravidla.

Tyto moduly je možné i za běhu clusteru jednoduše přepínat. Dále OpenShift dodatečně podporuje k8s CNI pluginy od třetích stran – jako rozhraní mezi OpenShiftem a k8s. Současné podporované CNI pluginy jsou: Cisco ACI, Flannel, Contiv, NSX-T, Nuage (rozebírány v sekci 1.4.3) a Kuryr (rozebíráno v sekci 1.5.4) [13, 41].

1.6.5 HW požadavky a instalace

Minimální HW požadavky OpenShiftu se liší od zvoleného produktu. Placené produkty musí být nasazené pouze na OS linuxové distribuce RHEL a Atomic Host. V případě OKD je již možné jej nasadit na Linuxové Distribuce Ubuntu, CentOS, RHEL a Fedora. Minimální HW požadavky pro OpenShift popisuje tabulka 1.4, kde

velikosti disku na kontroléru a uzlech jsou určené pouze pro samotný OpenShift bez podů. V produkci je pak pro každých 1 000 podů požadováno navíc 1 CPU a 1,5 GB paměti pro kontrolér a u hostujících uzlů záleží na očekávané velikosti jejich zátěže + 10 % režie navíc [41].

Tab. 1.4: Minimální HW požadavky uzlů OpenShiftu [41]

Uzly	**CPUs**	**RAM**	**Disk**
Kontrolér	4	16 GB	42+ GB
Uzel infrastruktury	1	8 GB	32+ GB
Výpočetní uzel	1	8 GB	32+ GB
Externí Etcd uzel	2	8 GB	20 GB SSD

Stejně jako OpenStack, OpenShift taktéž nabízí 2 druhy instalací – kontejnerizovaná, nebo pomocí balíčkovacího systému přímo na systém. Obecně jsou všechny instalace prováděny automatizačním nástrojem Ansible postupným spuštěním dvou playbooků [41]:

1. **prerequisites.yml**,
2. **deploy_cluster.yml**.

Kromě instalace čistého OpenShiftu, vytvořil RedHat i playbooky pro nasazení OpenStacku na OpenShift, nebo obráceně – OpenShift na OpenStack [41].

Pro testovací nebo učící účely lze nainstalovat Minishift, velmi jednoduchým způsobem jako Minikube. Vytvořila jej komunita k8s, proto i syntaxe jejich příkazů je shodná [40].

1.7 Výstup teoretické části

Cílem teoretické části je vybrání platformy, která nejlépe splňuje požadavky pro budování virtuální infrastruktury. V rámci této práce by zvolená platforma měla být schopna:

1. Vytvářet virtualizované sítě.
2. Vytvářet směrovače a připojit k nim libovolně zvolené virtuální sítě.
3. Vytvářet firewally, taktéž s libovolným umístěním.
4. Vytvářet kontejnery z Docker obrazů, připojit je do libovolně zvolených virtuálních sítí a podle toho automaticky nastavit potřebné síťování pro komunikaci s dalšími kontejnery.
5. Vytvářet kontejnery s podporou sandboxingu.
6. Rozdělit nasazené struktury do projektů (multi-tenancy).

Dle požadavků byla teoretická část platforem zaměřena na jejich základní strukturu, funkčnost a právě na uvedené aspekty výše. Pro budoucí účely byly srovnány i možnosti instalace s minimálními HW nároky. Následuje shrnutí výhod a nevýhod jednotlivých platforem cloud computingu.

Kubernetes

Výhodou k8s je jeho jednoduchost a tedy kratší čas potřebný k seznámení se s technologií. Dokumentace nejsou tak rozsáhlé jako u OpenStacku/OpenShiftu a stejně i tak pro svoji činnost nepotřebuje tolik komponent – z toho vyplývá i jednodušší hledání problémů (angl. Troubleshooting) při výskytu neobvyklé chyby. Na svých oficiálních stránkách nabízí také online interaktivní tutoriály, které nováčky postupně seznamují s k8s – ovládáním Minikube clusteru. K8s nasazuje aplikace do cloudu v podobě podů. Pody obsahují 1 nebo více kontejnerů, které mohou být vytvořeny na základě kontejnerových obrazů.

Značnou nevýhodou pro začátečníky může představovat potřeba naučit se strukturu a syntaxi YAML souborů (nebo JSON souborů, v tomto případě jsou ale používány podstatně méně), jelikož ne všechny potřebné úkony je možné vytvářet pomocí webového GUI. Také spousta manuálů je nejvíce interpretována jen YAML soubory. Pro plnou podporu sandboxingu je nutno doinstalovat další nástroj. Síťování je řešeno doinstalováním CNI pluginu ze třetí strany, ale bohužel žádný z prozkoumaných pluginů neumožňuje vytvářet směrovače volně ve virtuální infrastruktuře a za nimi další virtuální sítě – funkce směrovače je splňována jen na vstupu/výstupu každého uzlu. Podobné omezení platí i pro firewally – nastavovat pravidla pro příchozí nebo odchozí provoz lze jen pro skupiny podů. K8s neumožňuje vytvářet projekty – v základu celý k8s cluster je jeden velký jmenný prostor, ve kterém je ale možné vytvářet, jako náhradu za vytváření projektů, další jmenné prostory. Pokud ale vznikne potřeba nastavit více jmenným prostorům stejná přístupová práva, bude nutné nastavit všechny jmenné prostory manuálně zvlášť.

OpenStack

OpenStack splňuje v závislosti na zvolených službách všechny požadavky a mnohem více (např. vytváření virtuálních strojů). Služby zajišťující kontejnerizaci jsou Magnum a Zun. Magnum umožňuje automatizovat nasazení clusteru k8s do virtuálních strojů včetně CNI pluginu atd. Celou topologii i s vytvořenými kontejnery (pody) vykresluje v Horizonu. Zun umožňuje vytvářet kontejnery/kapsle (obdoba podů) bez potřeby vytvářet další cluster. Neutron plně podporuje vytváření směrovačů nebo firewallů kdekoliv v infrastruktuře. Keystone podporuje pro oddělení infrastruktury dvě úrovně – v rámci projektů; a nebo o další úroveň výš – v rámci domén.

Nevýhodou OpenStacku jsou jeho rozsáhlé dokumentace, kterým je pro jeho správné nasazení a nastavení potřeba rozumět. Taktéž pro hledání chyb je nutné porozumět alespoň základním službám a funkcionalitě OpenStacku. Pro podmínky sandboxingu je potřeba doinstalovat Kata kontejnery na výpočetní uzly, nebo zvolit nejnovější verzi OpenStacku – Ussuri. Kapsle lze ve službě Zun deklarovat jen pomocí YAML souborů.

OpenShift

OpenShift je postaven na platformě k8s a vylepšuje jeho vlastnosti a nedokonalosti. Základní nástroje (komunitní projekty), které by normálně v k8s bylo potřeba doinstalovávat do clusteru, OpenShift již obsahuje. Zlepšuje možnosti nasazování kontejnerů a již v základu poskytuje SDN síťování pro pody. Pro podporu sandboxingu platí stejné podmínky jako u k8s. Nabízí sofistikovanější rozdělení rolí uzlů pro směrování síťového provozu / nasazování podů. Funkce firewallu může být poskytována SDN modulem `ovs-networkpolicy` pro projekty; a CNI pluginem s podporou síťové politiky. OpenShift nabízí propracovanější webové rozhraní, přes které je možné i přidávat další výpočetní uzly do clusteru. To například značně ulehčuje práci pro klienty, kteří mají umístěný cluster na veřejném cloudu. Lze kombinovat i s OpenStackem.

Stejně jako v k8s – směrovače není možné volně vytvářet a budovat tak další infrastrukturu; a pro podporu sandboxingu je nutno doinstalovat další nástroj. Pro práci s OpenShiftem je minimálně požadována znalost Kubernetes.

1.7.1 Shrnutí a výběr platformy

Všechny platformy lze instalovat za pomocí Ansible playbooků, což značně ulehčuje práci při nasazení na servery.

K8s a OpenShift primárně slouží k nasazování aplikací a služeb do cloudu v kontejnerizované podobě a k jejich následné správě. Umožňují vytvářet repliky podů a následně mezi nimi rovnoměrně rozdělovat provozní zátěž, nebo poskytovat redundanci v případě výpadku jednoho výpočetního uzlu. V k8s (tím pádem i v OpenShiftu) běží kontrolní smyčky, které sledují stav běžících kontejnerů, v případě poruchy/zastavení jej restartuje, nebo v případě výpadku celého uzlu jej vytvoří na jiný uzel tak, aby vždy seděl počet vytvořených replik.

OpenStack naopak plně podporuje budování virtualizované infrastruktury, kterou může ale i nemusí sdílet s vybranými projekty. Vytvořený projekt může posloužit k vytváření vlastních scénářů, nebo výzkumným činnostem, kdy po splnění svého účelu budou jednoduše smazány a uvolněné výpočetní zdroje použity na jiné projekty.

Na základě získaných znalostí je sestavena tabulka 1.5 popisující splnění požadavků z pohledu jednotlivých platforem. Dříve uvedené podmínky nejvíce splňuje platforma OpenStack. Dalo by se považovat, že i s OpenShiftem lze splnit všechny požadavky, ale to jen za předpokladu nasazení společně s OpenStackem. Pro správu clusteru by ale bylo potřeba pochopit funkcionalitu všech tří platforem. Proto bude ve finále pro účely této diplomové práce použita platforma OpenStack.

Tab. 1.5: Platformy splňující požadavky uvedené na začátku kapitoly 1.7

Podmínka P:x	**Kubernetes**	**OpenStack**	**OpenShift**
P:1 sítě	+*	+	+
P:2 směrovače	−	+	−
P:3 firewally	+*	+	+
P:4 kontejnery	+	+	+
P:5 sandboxing	+*	+*	+*
P:6 dělení projektů	−	+	+

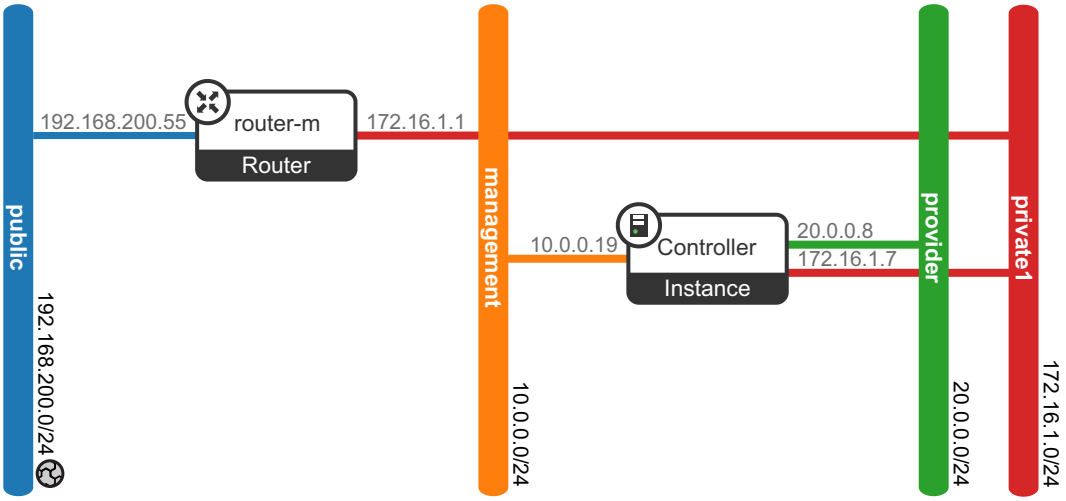
Pozn.: + umožňuje; +* umožňuje při doinstalování; − neumožňuje

2 Vlastní nasazení platformy OpenStack

Tato kapitola je věnována nasazení OpenStacku na VM a následnému testování funkčnosti jeho schopností. Ještě předtím je popsána předpříprava prostředí a diskového obrazu pro VM s jeho HW parametry.

2.1 Předpříprava prostředí a VM

Po domluvě s vedoucím práce je pro testování nasazen OpenStack v módu *all-in-one*. Pro nasazení je tedy vytvořena topologie dle obrázku 2.1 s jedním VM, jedním virtuálním směrovačem, jednou sítí a třemi virtuálními sítěmi. Síť `public` poskytuje připojení do Internetu. Virtuální směrovač `router-m` natuje provoz ze sítě `private1` do sítě `public`. V případě režimu `multinode` je zapotřebí nastavit síťování mezi všemi uzly v ideálním případě alespoň s rychlostí 1 GB/s.



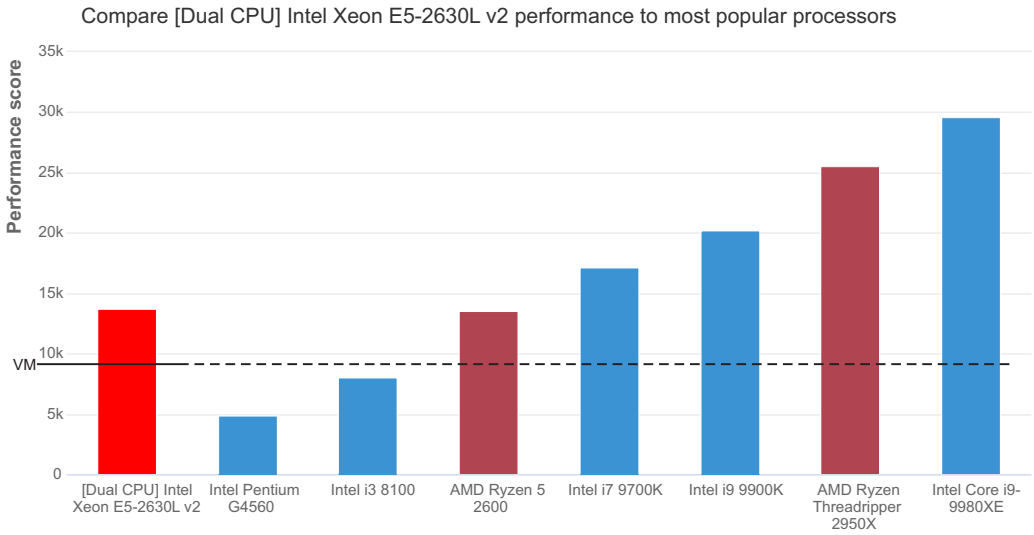
Obr. 2.1: Předpřipravená topologie

K vytvoření VM je použit cloudový obraz `CentOS-7-x86_64-GenericCloud-1907.qcow2` (volně dostupný na webu cloud.centos.org[1]), který je dále upraven pomocí nástrojů balíčku `libguestfs` (konkrétně `guestmount` a `virt-cutomize`). Jde především o vytvoření a přidání síťových skriptů (aby se při každém spuštění VM automaticky zapnuly rozhraní a zažádaly si o IP adresu DHCP serveru), úprava SSH serveru (vypnutí DNS a povolení přihlášení za uživatele root), autorizace přístupu osobního PC veřejným klíčem, nastavení proxy prostředí a další užitečné konfigurace ulehčující počáteční práci po vytvoření VM. Nastavený virtuální stroj má tyto parametry:

[1] `https://cloud.centos.org/centos/7/images/`

- **OS** CentOS Linux verze 7.6.1810 (Core),
- **výpočetní síla hostitele**: 2 procesory [Dual CPU] Intel Xeon CPU E5-2630L v2 @ 2,40 GHz (2 × 6 fyzických CPU jader, 2 vlákna na jedno jádro) virtualizován na model: Intel Xeon E3-12xx v2 (Ivy Bridge, IBRS) – cca 8 z 24 logických CPU s podporou KVM-intel virtualizace, bez integrované grafické karty,
- **RAM** DDR3 @ 1,60 GHz o velikosti 2 × 16 GiB,
- **HDD** o velikosti 204 GiB,
- 3 **síťová rozhraní** – jedno pro vzdálený přístup a Internet, dvě pro sítě managementu a poskytovatele.

Pro lepší orientaci ve výkonu VM je výkon jednoho procesoru hostitele porovnán s aktuálně nejpopulárnějšími procesory na obrázku 2.2 (výkon VM odpovídá 2/3 jednoho procesoru).

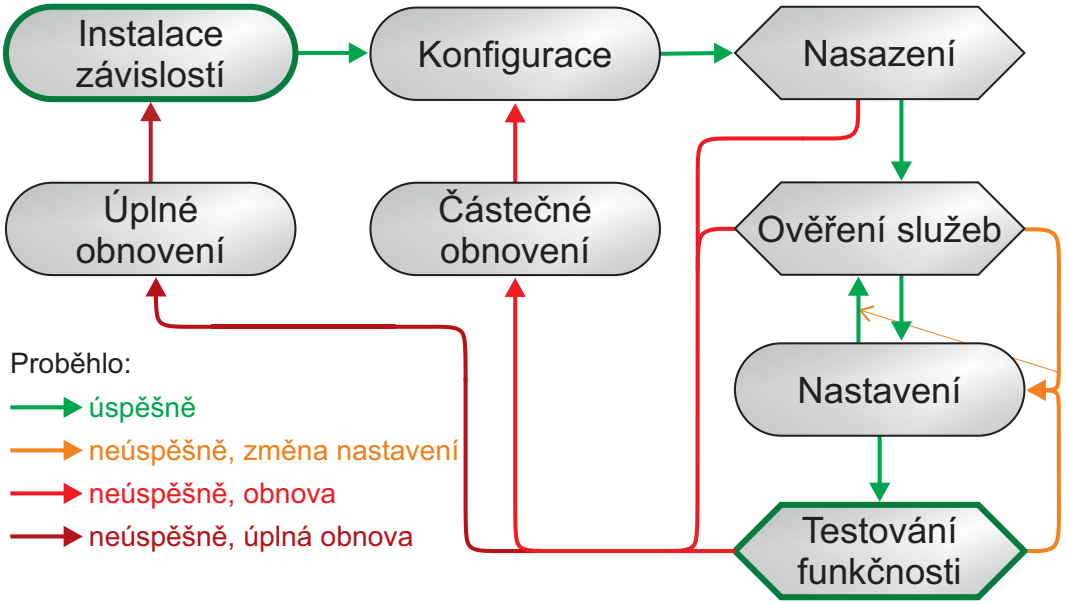


Obr. 2.2: Porovnání procesoru hostitele s nejpopulárnějšími procesory, včetně vyznačení výkonu VM (převzato a upraveno z [44])

2.2 Instalace

K instalaci OpenStacku je vybrán nástroj Kolla a Kolla-ansible. Pro splnění všech požadavků v kap. 1.7 jsou vybrány následující služby OpenStacku: Keystone, Nova, Glance, Placement, Neutron, plugin Neutron-FWaaS, Horizon (včetně pluginů použitých služeb), Cinder, Cinder-backend-lvm, Zun, Kuryr a plugin Etcd.

Instalace s následným zprovozněním OpenStacku je prováděna dle obrázku 2.3. Po instalaci závislostí pro kollu-ansible a nastavení konfiguračních souborů je OpenStack nasazován ansible playbooky. Následuje vždy ověření správné funkčnosti všech služeb, donastavení administrátorem a v poslední řadě testování funkčnosti OpenStacku jeho používáním (znázorněno jako zelená signalizace). V případě, že je během ověřování funkčnosti služeb nebo testování nalezena chyba (oranžová signalizace), administrátor se vrací ke kroku nastavení služeb a snaží se najít příčinu. Po změně nastavení procesy opakuje obvykle až do doby, než najde příčinu. Příčina je ale mnohdy již ve špatné konfiguraci služeb před nasazením, proto je potřeba OpenStack kompletně vymazat (všechny jeho kontejnery, volumy, obrazy, způsobená nastavení – světle červená signalizace) a po změně konfigurace cyklus opakovat. Pokud i přesto nebudou všechny služby OpenStacku zprovozněny, nezbývá než obnovit VM pomocí snapshotu, nebo začít od začátku (tmavě červená signalizace). Je dost možné, že během opakovaných instalací, v systému přetrvávají nějaké pozůstalé konfigurace, které narušují činnosti nového nasazení.



Obr. 2.3: Stavový diagram instalace a zprovoznění OpenStacku pomocí nástroje kolla-ansible

2.3 Testování funkčnosti

Po úspěšném nasazení OpenStacku byly zkoušeny různé příkazy k ovládání OpenStacku za účelem ověření funkčnosti a vyzkoušení schopností jednotlivých služeb. Následuje nejdůležitější výčet:

- **Keystone** – vytvoření nového projektu s novým uživatelem, definice veřejného klíče uživatele, nastavování kvót projektů a práv uživatelů.
- **Glance** – uložení diskového obrazu a vytvoření flavoru (charakteristika parametrů VM).
- **Neutron** – Vytvoření externí a interní sítě, propojení směrovačem(a), konfigurace nových směrovacích cest, uvolnění plovoucích IP adres, definice bezpečnostních skupin.
  - **FWaaS** – definice různých pravidel, příchozí a odchozí politiky a firewallu s aplikací na úrovni portů a směrovačů se sledováním změn ve stavovém firewallu směrovače.
- **Nova** – Vytvoření VM, připojení do interní sítě, přiřazení plovoucí IP, nastavení proxy a otestování dostupnosti Internetu. Vytvoření snapshotu s následnou rekonstrukcí.
- **Zun** – vytvoření kontejneru s implementací běhového prostředí kata-runtime, vytvoření nového obrazu na základě provedených změn v kontejneru s následnou rekonstrukcí.
- **Kuryr** – propojení kontejneru se síťovou topologií Neutronu, ověření DHCP a internetu v kontejneru.
- **Cinder** – vytvoření volumu, připojení k VM, připojení ke kontejneru.
- **Horizon** – vyzkoušení funkčnosti výše zmíněných služeb ve webovém rozhraní.
- **Heat** – vytvoření velmi jednoduché šablony.
- **Placement** – slouží ostatním službám, pokud ostatní fungují v pořádku, platí to samé i pro Placement.

2.4 Manuál k OpenStacku

V rámci práce byl vypracován komplexní manuál k OpenStacku s detailním popisem jednotlivých procesů obrázku 2.3 včetně problémů, se kterými se autor práce v průběhu nasazení musel vypořádat. Manuál k OpenStacku, který je taky vlastním přínosem práce, byl po konzultaci s vedoucím práce odevzdán mimo hlavní práci a to z důvodu velkého rozsahu dalších 44 stran. Ukázka manuálu je dostupná v příloze B.

3 Experimentální ověření náročnosti OpenStacku

V rámci měření jsou vytvořeny scénáře s cílem zjistit, jak hodně je daný scénář závislý na HW zdrojích jako jsou: disková kapacita, CPU, paměť RAM nebo vytěžování sítě. Testovací scénáře jsou rozděleny do tří skupin:

1. skupina se zaměřením na konkrétní úkony v OpenStacku,
2. skupina s pokročilejším scénářem a zátěžovými testy,
3. skupina se čtyřmi scénáři, které do jisté míry kombinují první a druhou skupinu – scénáře jsou rozděleny do menších celků s propracovanější simulací zátěže.

V rámci testování náročnosti je vytvořeno celkem 41 skriptů převážně v jazyce Bash pro: skenování vytíženosti systému (3); vytváření a mazání scénářů (19); simulaci zátěže (5+1); automatizaci (9); a transformaci surových dat do čitelného/použitelného stavu (4). U pokročilejších scénářů jsou vytvořeny i vlastní kontejnerové obrazy (4) přizpůsobené potřebám testování. Všechny soubory jsou dostupné v příloze.

Terminologie

V následujících částech jsou používány termíny:

- Sxx – velké „S“ značí scénář a xx je číslem scénáře.
- sxx – malé „s“ značí skript nebo simulaci zátěže scénáře. Každý scénář Sxx je vykonáván svým vlastním skriptem sxx.

3.1 První skupina scénářů

První skupina scénářů se zaměřuje na náročnost dílčích příkazů (malých scénářů) OpenStacku, aby se dalo efektivně oddělit, jaké procesy mají největší vliv na zátěž HW. Pro měření je vytvořen jednoduchý skript `s00v3`[1] (výpis 3.1), který vypisuje průměrné vytížení CPU za 1, 5 a 15 minut (pro měření používán nejdelší průměr), využití disku (celkové a v rámci logů OpenStacku) a paměti RAM. Skript měření je vykonán vždy před spuštěním skriptu scénáře a 15 minut po jeho dokončení.

Výpis 3.1: Skript pro měření aktuálního vytížení systému

```
#!/bin/bash
echo "CPU: $(cat /proc/loadavg | cut -d' ' -f1-3)"
free -m | grep Mem | awk '{printf ("Mem used: %d MiB \n"),$3}'
df -m | grep vda1 | awk '{printf ("Disk: %d MiB \n"),$3}'
```

[1] Celkově jsou pro měření vytvořeny 3 skripty: `s00v3` s výpisem na obrazovku, `s00v2` s výpisem do souboru a `s00` pro výpis do souboru, navíc s průměrným vytížením sítě za 5 minut.

```
echo "Logs: $(du -sm /var/log/kolla/ | awk '{print $1}') MiB"
```

Jednotlivé scénáře jsou rozepsány následovně:

1. Vytvoření virtuální privátní sítě 10.0.10x.0/24.
2. Vytvoření virtuálního směrovače.
3. Připojení sítě do směrovače.
4. Vytvoření firewall pravidla – povol TCP port „100x“.
5. Vytvoření nové síťové politiky.
6. Přidání pravidla do síťové politiky.
7. Vytvoření firewall skupiny.
8. Přidání pravidla jako příchozí do firewallu.
9. Aplikování firewallu na port směrovače s IP 10.0.10x.1.
10. Vytvoření nového projektu.
11. Vytvoření nového uživatele, přiřazení do projektu a přidělení role `_member_`.
12. Vytvoření VM s obrazem `cirros` (o velikost 12,13 MB), 512 MB paměti RAM, 1 CPU jádrem, 1 GB diskem a připojením do privátní sítě.
13. Vytvoření kontejneru v běhovém prostředí `runc` se stejnými parametry jako VM až na: použití obrazu `cirros` pro kontejnery (o velikosti 10,3 MB) a bez omezení úložiště.
14. Vytvoření kontejneru (sandboxu) v běhovém prostředí `kata-runtime` se stejnými parametry jako předešlý kontejner.

Jelikož se využití HW prostředků neustále mění, je potřeba počítat s malou nejistotou. Pro přesnější měření budou tohoto důvodu scénáře S1–S11 opakovány 10krát a scénáře S12–S14 opakovány 6krát (VM může použít nejméně jedno CPU, zbylé dvě jsou minimální podmínkou pro kontrolér). Výsledně naměřené hodnoty jsou zprůměrovány dle počtu opakování.

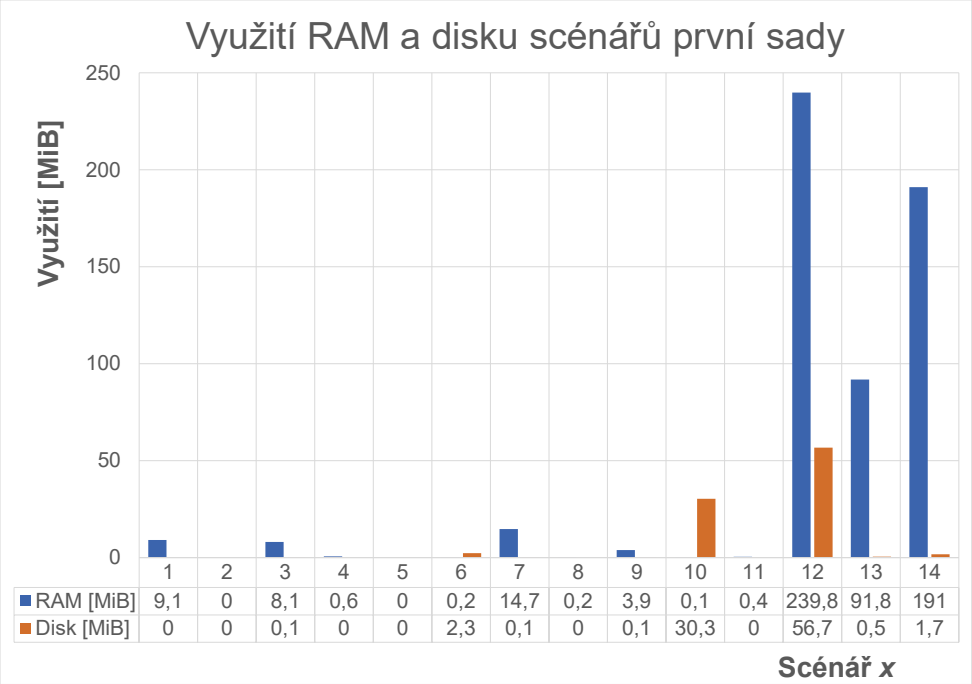
3.1.1 Výsledky měření

Výsledkem měření je tabulka 3.1 (graf 3.1) s průměrným využití HW zdrojů na operace jednoho scénáře. Vliv scénářů na zatížení CPU se nedá jednoznačně určit. Výsledky jsou hodně ovlivňovány akcemi OpenStacku prováděnými v různých intervalech (stabilnější výsledek byl zaznamenán jen u posledních tří scénářů), viz graf 3.2.

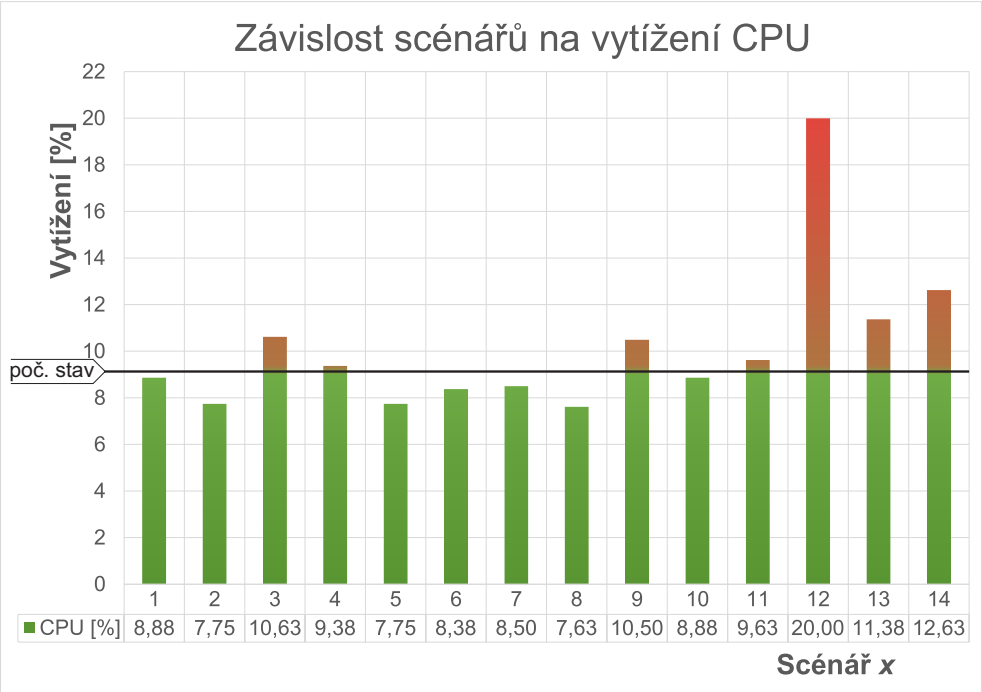
Dle očekávání jsou na využití HW zdrojů nejvíce závislé VMs a naopak nejméně kontejnery. Sandboxové kontejnery jsou kombinací obou technologií, tudíž i jejich využití zdrojů je někde na pomezí.

Tab. 3.1: Využití HW zdrojů příkazy OpenStacku

Scénář S:x	**RAM [MiB]**	**Disk [MiB]**	**CPU [%]**
S:1 síť	9,1	≈ 0	nelze určit
S:2 směrovač	≈ 0	≈ 0	nelze určit
S:3 připojení sítě	8,1	0,1	nelze určit
S:4 pravidlo	0,6	≈ 0	nelze určit
S:5 síťová politika	≈ 0	≈ 0	nelze určit
S:6 přidání pravidla	≈ 0	2,3	nelze určit
S:7 firewall	14,7	0,1	nelze určit
S:8 přidání pol. do f.	≈ 0	≈ 0	nelze určit
S:9 aplikace firewallu	3,9	0,1	nelze určit
S:10 projekt	≈ 0	30,3	nelze určit
S:11 uživatel	0,4	≈ 0	nelze určit
S:12 VM	239,8	56,7	$\approx 1,63$
S:13 kontejner	91,8	0,5	$\approx 0,19$
S:14 sandbox	191	1,7	$\approx 0,40$



Obr. 3.1: Graf vytížení RAM a disku scénářů první sady



Obr. 3.2: Graf závislosti CPU na běh jednotlivých scénářů (nezprůměrováno)

3.2 Druhá skupina scénářů

Tato skupina je zaměřená na vytvoření praktického scénáře S15 s následným spouštěním zátěžových testů `s16`. Praktický scénář obsahuje:

- 2 privátní sítě 10.0.x.0/25 a 10.0.x.128/25 propojené jedním směrovačem,
- 2 kata kontejnery v roli klienta (pro HTTP a FTP),
- 2 kata kontejnery v roli serveru (HTTP a FTP),
- 1 kata kontejner v roli útočníka (obraz `parrotsec/security`), ale v zátěžových testech nakonec nebyl využit.

Kontejnerový obraz HTTP serveru je modifikován z `nginx:latest` přidáním vlastní úvodní stránky `index.html` o velikosti 100 kB. Obraz FTP serveru je rovněž modifikován a to z `stilliard/docker-pure-ftpd`: nastavení FTP serveru, vytvoření uživatele, vložení databáze jmen o velikosti 37,2 MB. Pro klienty byl vytvořen obraz `xstodu07/photon`, který do základního obrazu `photon:latest` přidává jednoduché zátěžové skripty:

- `http.sh` – stahování zdroj. kódu uvítací stránky HTTP serveru (výpis 3.3).
- `ftp.sh` – stahování databázového souboru každých 10 s (výpis 3.3).

Všechny soubory potřebné k vytvoření vlastních kontejnerových obrazů jsou dostupné v příloze.

Výpis 3.2: Skript pro stahování zdrojového kódu stránky

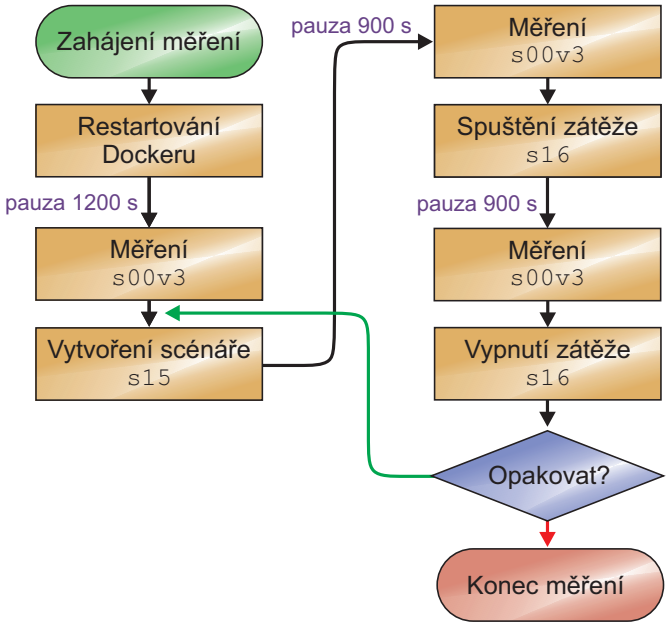
```
#!/bin/bash
while true; do
  curl 10.0.${i}.201:80
  sleep 10
done
```

Výpis 3.3: Skript pro stahování souboru z FTP

```
#!/bin/bash
while true; do
  ftpget -g -u uzivatel -P uzivatel 10.0.${i}.202 names.txt
  sleep 10
done
```

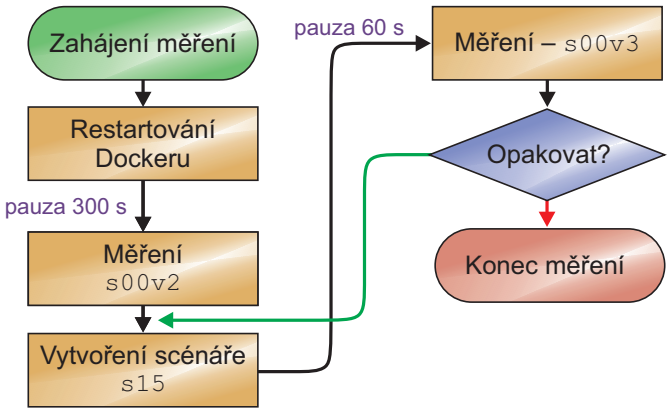
Skript `s16` spouští nebo vypíná patřičné zátěžové skripty na všech kontejnerech klientů v závislosti na jejich roli – FTP/HTTP. Postup měření znázorňuje obrázek 3.3. Spouštění jednotlivých kroků je prováděno manuálně, restartování Dockeru uvolní veškeré zdroje, které jsou používány službami OpenStacku. Před spuštěním skriptu měření je čekáno minimálně 15 minut pro získání průměrného vytížení CPU a sítě. Vytížení sítě je zaměřeno především na interní komunikaci OpenStacku, pomocí:

```
nload -t 1000 -a 300 -m lo
# -t: vzorek každých 1000 ms, -a: průměr za 300 s, -m: pro rozhraní loopback
```



Obr. 3.3: Vývojový diagram měření S15 se zátěží

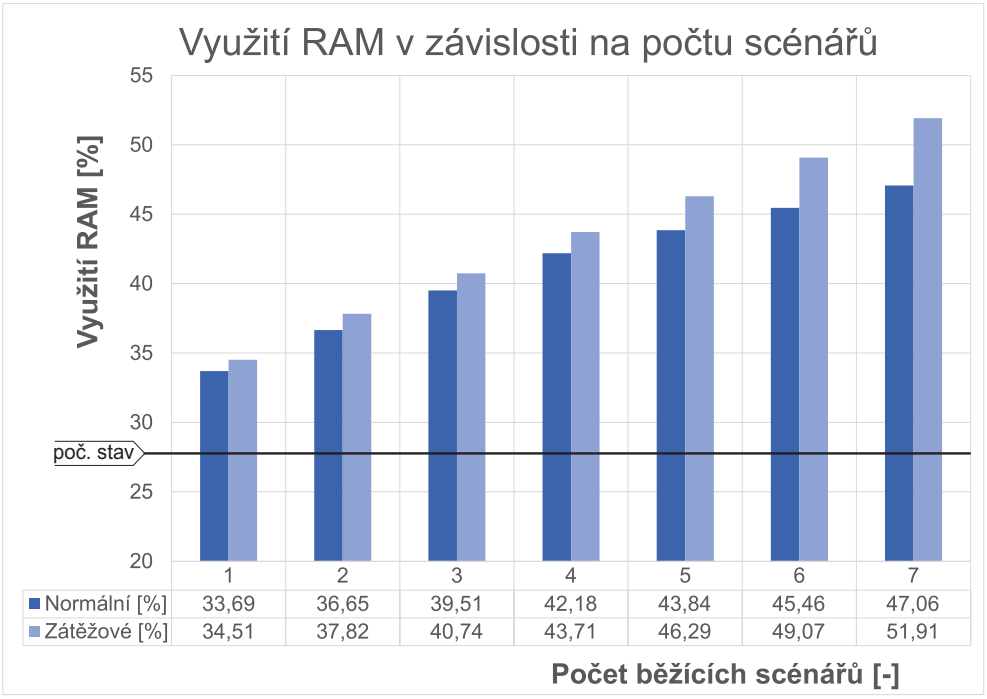
Dodatečně je vytvořen automatizační skript `s17` (obr. 3.4) se zaměřením na využití RAM a disku bez zátěžových testů. V předešlém postupu jsou tyto dvě veličiny velmi zkreslovány zátěžovými testy.



Obr. 3.4: Vývojový diagram měření S15 se zaměřením na RAM a disk bez zátěže

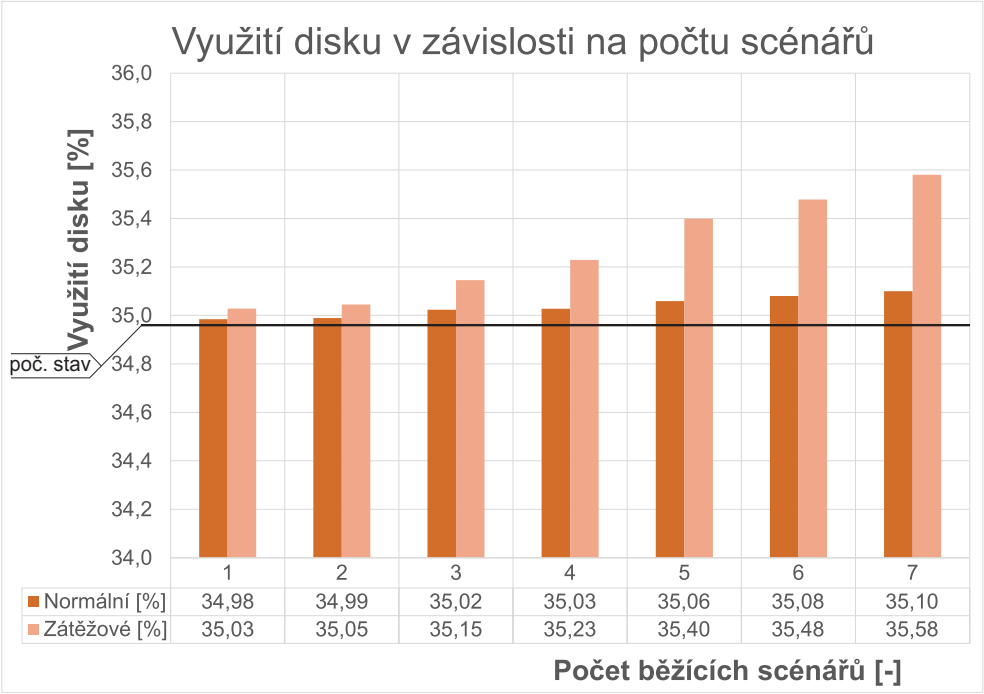
3.2.1 Výsledky měření

Výsledky měření jsou reprezentovány v grafické podobě pro: **RAM** (graf 3.5), **disk** (graf 3.6), **CPU** (graf 3.7) a vytížení **sítě** (graf 3.8) v rámci interní komunikace OpenStacku. Jelikož jsou služby OpenStacku restartovány, prvotní vytvoření scénáře má vyšší spotřebu paměti RAM.



Obr. 3.5: Graf vytížení RAM v závislosti na počtu scénářů

Využití disku je v rámci zátěžových testů vytěžováno především stáhnutím souboru z FTP serveru. V normálním stavu je disk nepatrně využíván zápisem do logů a SQL databází MariaDB.

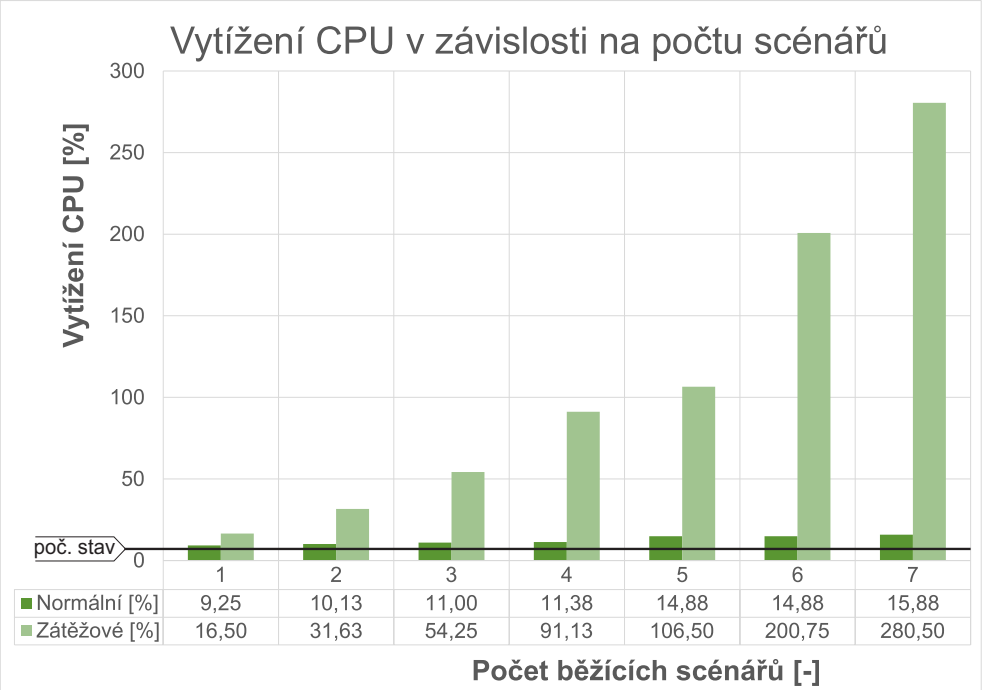


Obr. 3.6: Graf vytížení disku v závislosti na počtu scénářů

Na vytížení CPU se nejvíce podílelo stahování FTP souboru, které se výrazně začne projevovat už od druhého scénáře. Stahování 37,2MB souboru vícekrát zároveň je pak limitováno, jak rychlostí zápisu na disk, tak i výpočty v průběhu směrování datového toku sítěmi. Cesta datového toku je následující: `kontejner` > `tap` rozhraní > `vxlan` síť 1 > `tap` rozhraní > `bridge` směrovač > (`fyzické rozhraní`, pokud je cílový kontejner na jiném výpočetním uzlu) > `bridge` směrovač > `tap` rozhraní > `vxlan` síť 2 > cílový `kontejner`. Manuálně byl proveden výpočet **generování průměrné zátěže na 1 scénář** v případě zatížení:

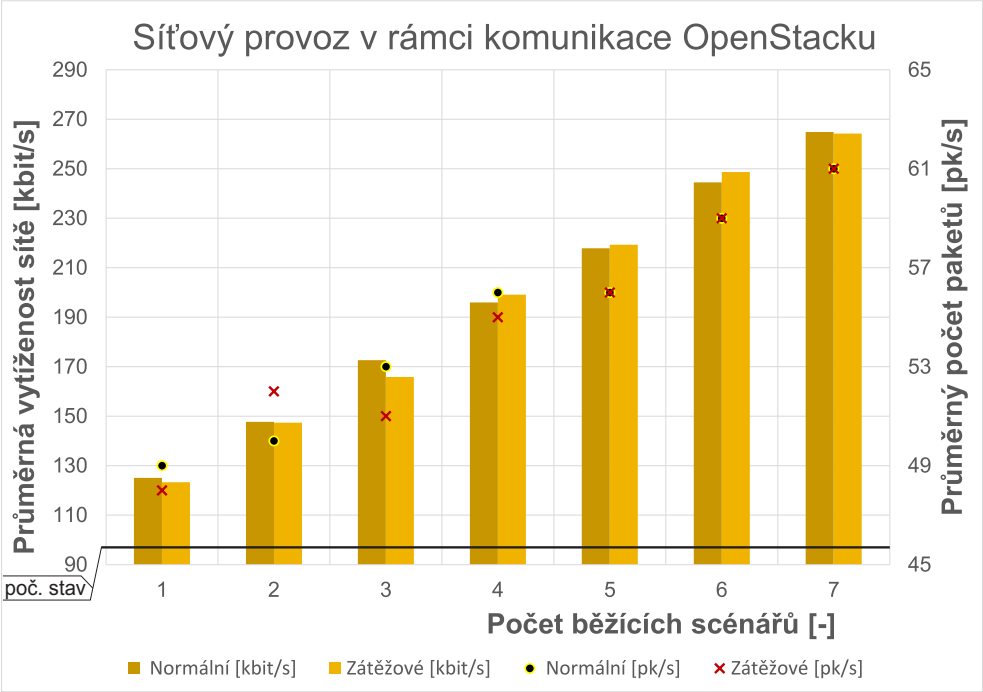
- **Jednoho** běžícího scénáře – 2,060 MB/s RX; 2,061 MB/s TX.
- **Sedmi** běžících scénářů – 0,881 MB/s RX; 0,855 MB/s TX.

I přes značné přetížení si OpenStack dokázal se zátěží poradit a na jednotlivých rozhraních nedocházelo k žádným chybám, kolizím nebo k zahazování paketů (kontrolováno pomocí `ifconfig | grep errors`). Místo toho se generování celkového provozu snížilo na 42,1 % předchozí rychlosti. Z důvodu vysokého přetížení procesoru (až o 2,8násobek) nebylo ve vykonávání dalších scénářů pokračováno.



Obr. 3.7: Graf vytížení CPU v závislosti na počtu scénářů

Vytvoření více scénářů má vliv na vyšší provozní zátěž OpenStacku. Simulace přetížení systému nemá na režijní provoz komunikace OpenStacku žádný vliv.



Obr. 3.8: Graf síťového provozu v rámci interní komunikace OpenStacku

3.3 Třetí skupina scénářů

Z předchozích měření není stále možné jednoznačně určit, jaký vliv na HW zdroje mají jednotlivé scénáře při zátěži. Předchozí scénář je proto rozdělen do čtyř skupin:

1. S21 – vytvořit kata kontejner útočníka (klienta) s připojením do privátní sítě a jedním směrovačem.
2. S22 – stejné jako S21, navíc s připojeným web-serverem.
3. S23 – stejné jako S22, ale každý kata kontejner je připojený do vlastní privátní sítě propojené směrovačem.
4. S24 – stejné jako S23, ze směrovače se ale nyní stává firewall.

Kontejnerový obraz útočníka je modifikován z `kalilinux/kali-rolling` instalací `metasploit-framework` a balíčku nástrojů `kali-linux-top10` (obsahuje i `nmap`). Dále je v Kali Linuxu nainstalován a zprovozněn nástroj `phantomjs` (pro simulaci webového prohlížeče). V neposlední řadě je do obrazu zkopírován: `skript.sh` ke spouštění tří druhů zátěže (`nmap`, `curl`, `phantomjs`), `phscript.js` k definici zátěže pro `phantomjs` v jazyce javascript a poslední soubor `00save_info` plnící funkci malé lokální databáze. Pro web-server je použit kontejnerový obraz `webgoat/webgoat-8.0`. WebGoat je záměrně nezabezpečená aplikace, která umožňuje zkoušet různé zranitelnosti.

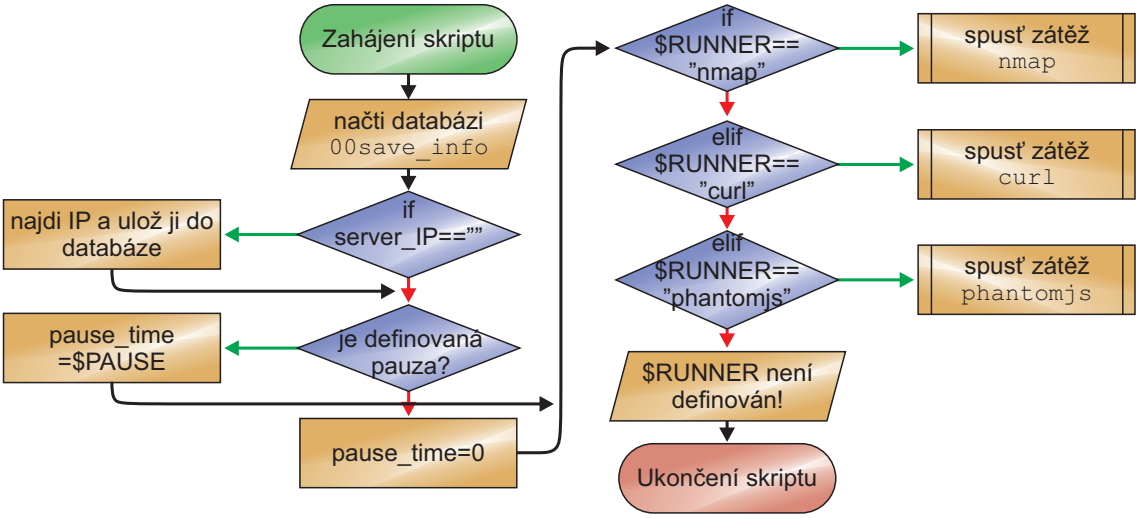
Veškeré vytváření scénářů a měření je nyní automatizováno skripty:

- `s31`–`s34` – automatizují vytváření scénářů (`s21`–`s24`), spouštění a zastavení simulací zátěže (`s20`), vše včetně zaznamenání dílčích měření (`s00`).
- `s35`–`s38` – zaměřují se na využití RAM a disku bez zátěžových testů, založené na stejném principu jako `s17`, viz obr. 3.4.

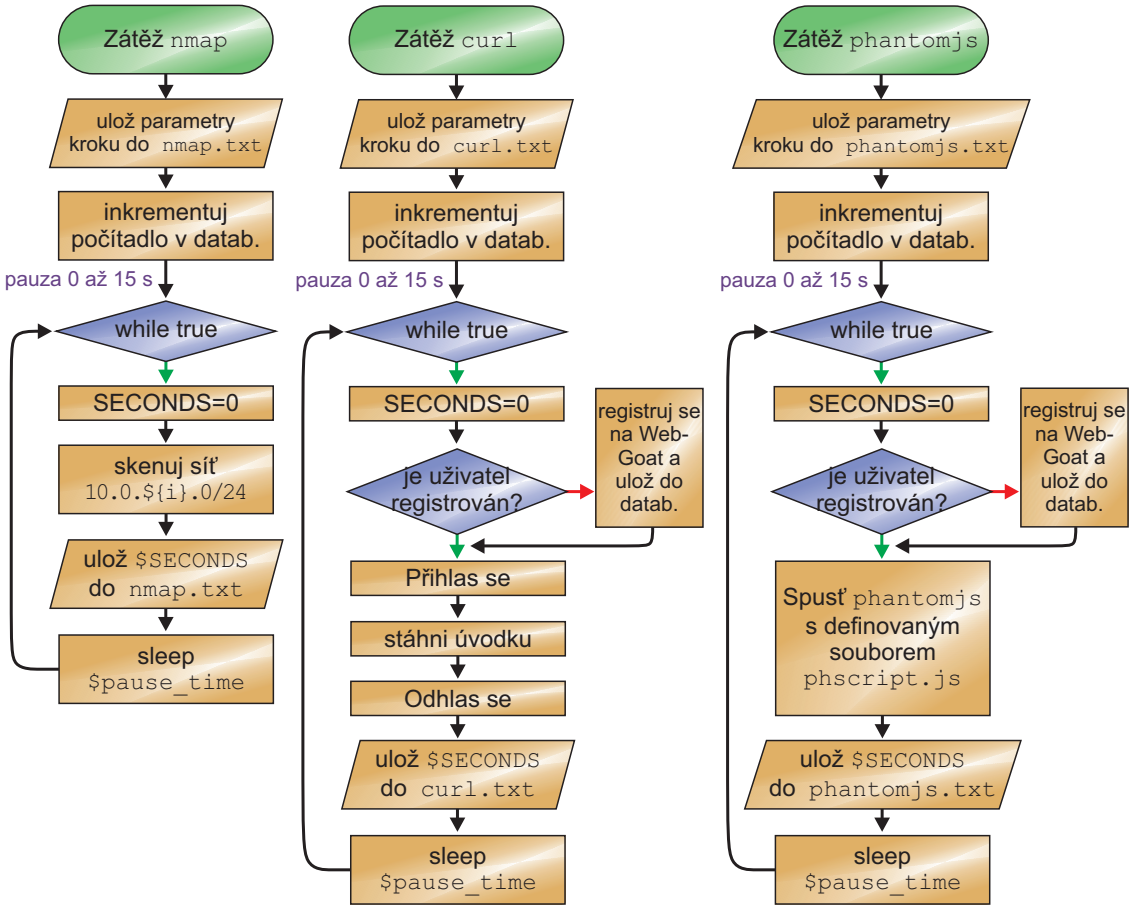
Skript `s20` spouští nebo vypíná na všech běžících kontejnerech Kali Linuxu skript `script.sh`. Spouští se příkazem:

```
./20-new-scenariox-workload <RUNNER> <PAUSE> <STOP>
# RUNNER - definuje typ spouštěné zátěže
# PAUSE - definuje pauzu mezi cykly zátěže v sekundách
# STOP - pokud se rovná "stop", zastaví skript v kontejnerech
```

Funkcionalitu skriptu `script.sh` popisuje obrázek 3.9. Jednotlivé kroky zátěží pak vysvětluje algoritmus na obr. 3.10. Platforma WebGoat je ale založená na kódu java, proto samotný `curl` nestáhne celý obsah stránky, ale jen její zdrojový kód v HTML. Z tohoto důvodu je přidán do zátěže kód `phscript.js` s podobnými kroky jako u `curlu`, napsaný v javascriptu. Kód je pak vykonáván aplikací `phantomjs`.



Obr. 3.9: Vývojový diagram skriptu `script.sh` simulujícího zátěž v Kali Linuxu



Obr. 3.10: Vývojový diagram kroků jednotlivých zátěží ve skriptu `script.sh`

Vykonávání automatizačních skriptů je prováděno například dle výpisu 3.4 (scénáře S22–S22 jsou prováděny i se zátěží pom. `curl` s 10s pauzou a `phantomjs` také s 10s pauzou mezi cykly). Po restartování Dockeru je počkáno 300 s a následně spuštěn skript měření `s00`, který samotný trvá taktéž 300 s. Vzhledem k tomu, že měření se stalo mnohem komplexnější, vytížení CPU je v těchto scénářích počítáno z pětiminutového průměru (i tak trvá takové měření pomocí `s31`: 3 h nebo `s32`: 7 h). Vylepšenou verzi skriptu měření (`s00`) zobrazuje výpis 3.5.

Výpis 3.4: Automatizace scénáře S21 skriptem `s31` včetně zátěže a měření

```
#!/bin/bash
SCENARIOS=10
sleep 300
./00-measurement.sh "1before"  # první mezi-výpočet před měřením

for scene_number in $(seq 1 ${SCENARIOS}); do
  echo "*********Scenario: ${scene_number} *************" >> result.txt
  ./21-new-scenario1-setup.sh ${scene_number}
  sleep 300  # pětiminutová dostatečná pauza, aby se všechny kontejnery vždy
    ↪vytvořily a žádné vnitřní procesy nezasahovaly do měření vytížení CPU
  ./00-measurement.sh "2after-21-before-nmap"
  ./20-new-scenario-workload.sh nmap 0
  sleep 60
  ./00-measurement.sh "3workload-nmap"
  ./20-new-scenario-workload.sh nmap 0 stop
  sleep 120  # 2krát delší pauza, aby se systém i přes vysokou zátěž stihl
    ↪vzpamatovat.
done
```

Výpis 3.5: Struktura skriptu měření `s00`

```
#!/bin/bash
echo "" >> result.txt
echo "_________________Step: $1 _________________" >> result.txt
echo "" >> result.txt
bwm-ng --type avg --avglength 300 --output plain --unit bytes --timeout 1000
  ↪--count 300 --ansiout | sed '/iface/h;//!H;$!d;x' >> result.txt

echo "************************************" >> result.txt
echo "CPU: $(cat /proc/loadavg | cut -d' ' -f1-3)" >> result.txt
free -m | grep Mem | awk '{printf \
("Mem used: %d MB \n\
Mem free: %d MB \n\
Mem shared: %d MB \n\
Mem buff/cache: %d MB \n\
Mem available: %d MB \n\
"),$3,$4,$5,$6,$7}'\
>> result.txt
df -m | grep vda1 | awk '{printf ("Disk: %d MB \n"),$3}' >> result.txt
echo "Logs: $(du -sm /var/log/kolla/ | awk '{print $1}') MB" >> result.txt
```

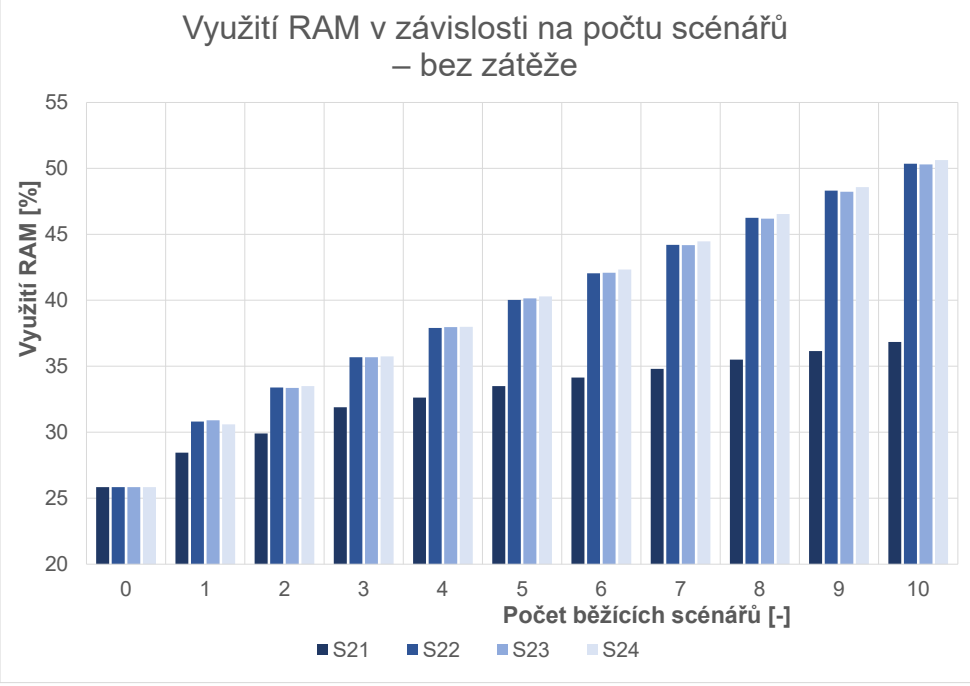
3.3.1 Výsledky měření

Z důvodu velkého množství surových dat byly vytvořeny skripty, které transformují soubory s naměřenými hodnotami do použitelné formy (shromažďují data stejných typů do sloupce pod sebe):

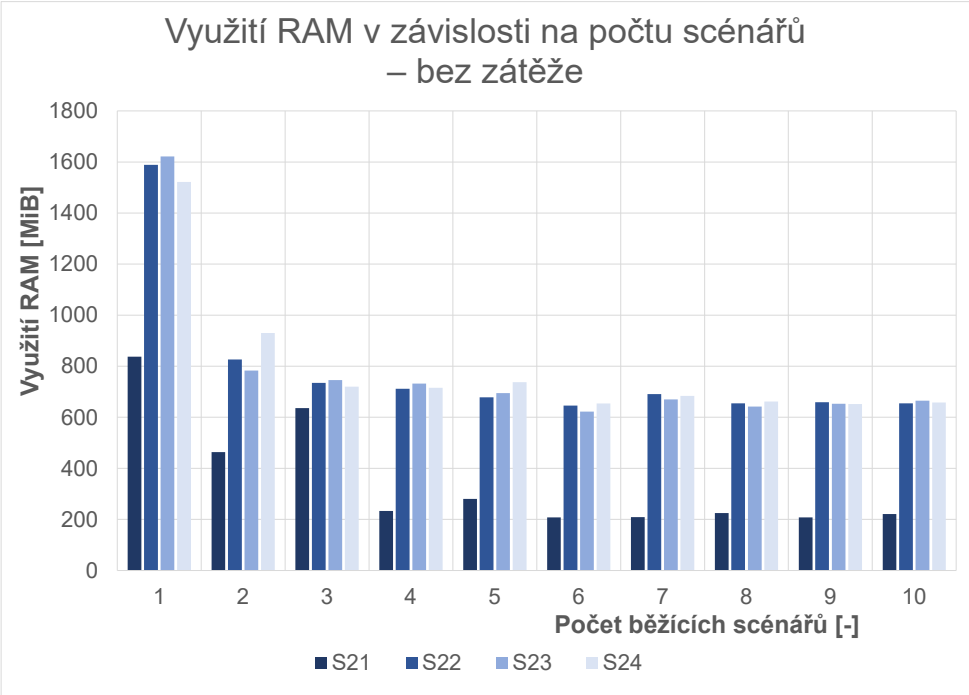
- `40-values-printer.sh` – transformace dat z měření scénáře S21.
- `40-values-printer2.sh` – transformace dat z měření scénářů S22–S24.
- `40-values-printer3.sh` – transformace dat pro RAM a disk z `s35`–`s38`.
- `41-avg-values.sh` – vypočítá průměrnou dobu trvání jednoho cyklu zátěže pro všech 10 opakování.

Prvním výstupem jsou grafy bez spuštěné zátěže pro RAM (graf 3.11 a s rozdílem před a po nasazení scénáře graf 3.12), disk (graf 3.13) a CPU (graf 3.14). „Nulté scénáře“ uvádí počáteční stav před vytvářením scénářů. Jejich hodnoty jsou vypočteny zprůměrováním počátečních stavů všech čtyřech scénářů. Výsledný rozdíl je přičten anebo odečten s naměřenými hodnotami opakovaných scénářů 1–10 (kromě CPU, zde je průměr počítán jen v rámci nultého scénáře).

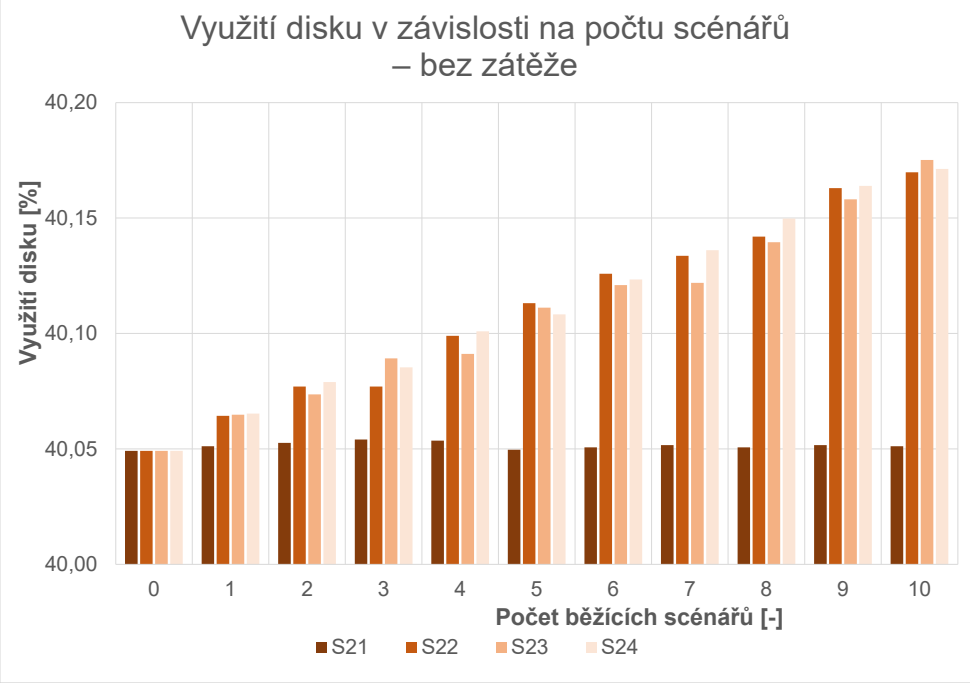
Následují výsledky simulací jednotlivých zátěží použitím nástrojů `nmap`, `curl` a `phantomjs`. Grafy využití RAM a disku znázorňují rozdíl ve spotřebě daných zdrojů v časech: před spuštěním zátěže a během spuštěné zátěže.



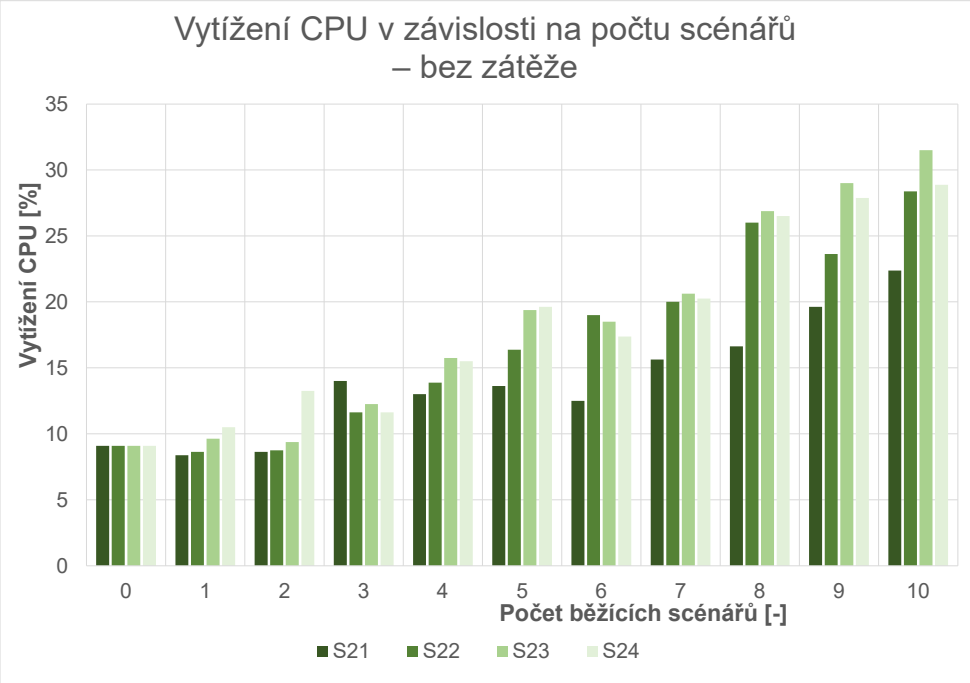
Obr. 3.11: Graf celkového využití RAM pro scénáře S21–S24 bez spuštěné zátěže v procentech



Obr. 3.12: Graf využití RAM pro scénáře S21–S24 bez spuštěné zátěže v MiB

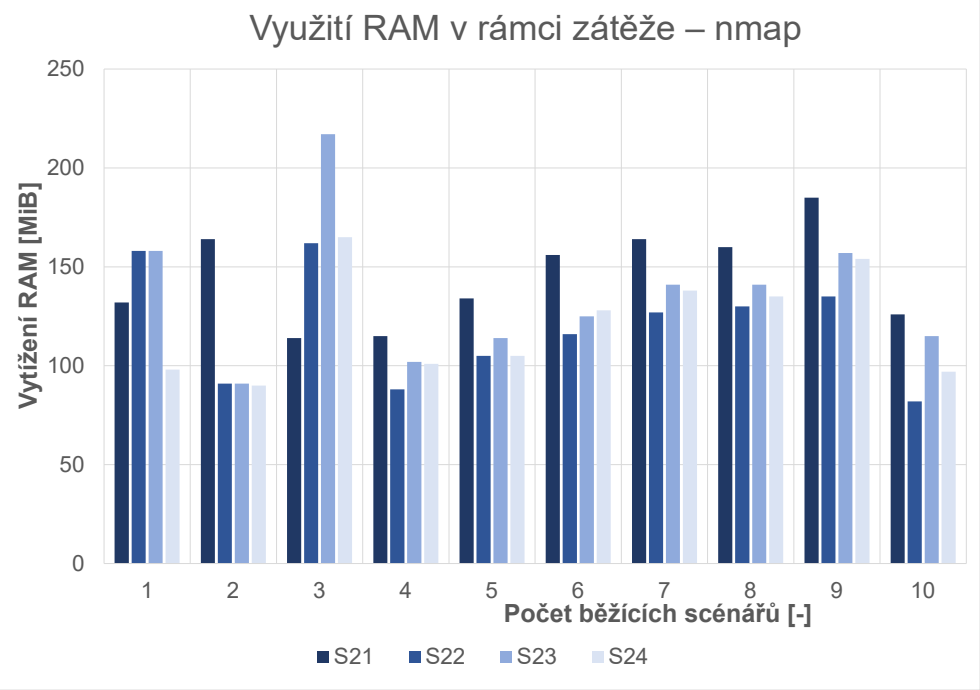


Obr. 3.13: Graf celkového využití disku pro scénáře S21–S24 bez spuštěné zátěže

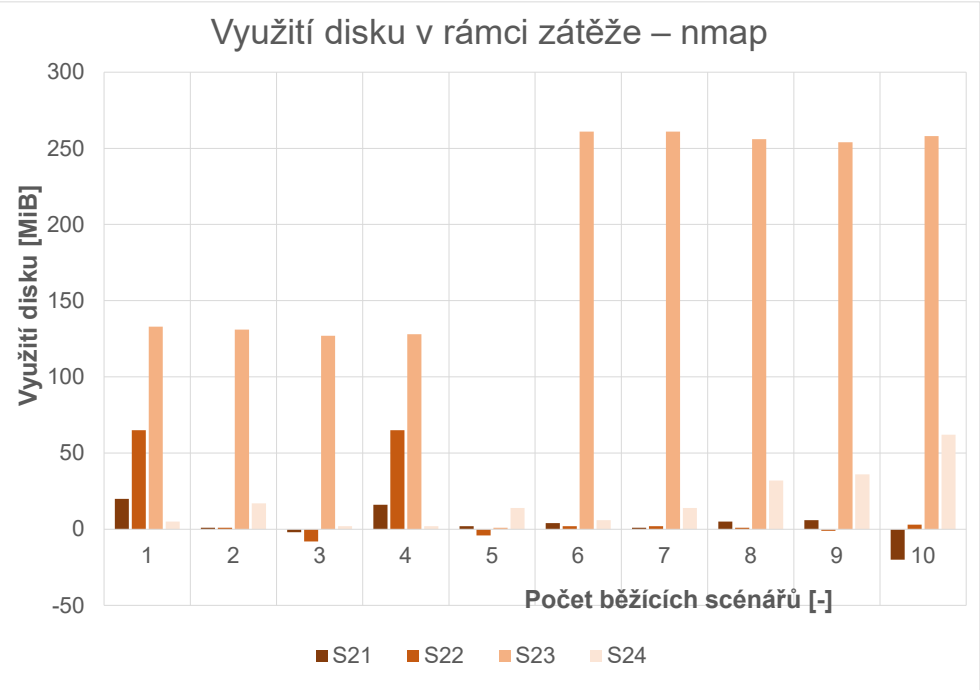


Obr. 3.14: Graf celkového vytížení CPU pro scénáře S21–S24 bez spuštěné zátěže

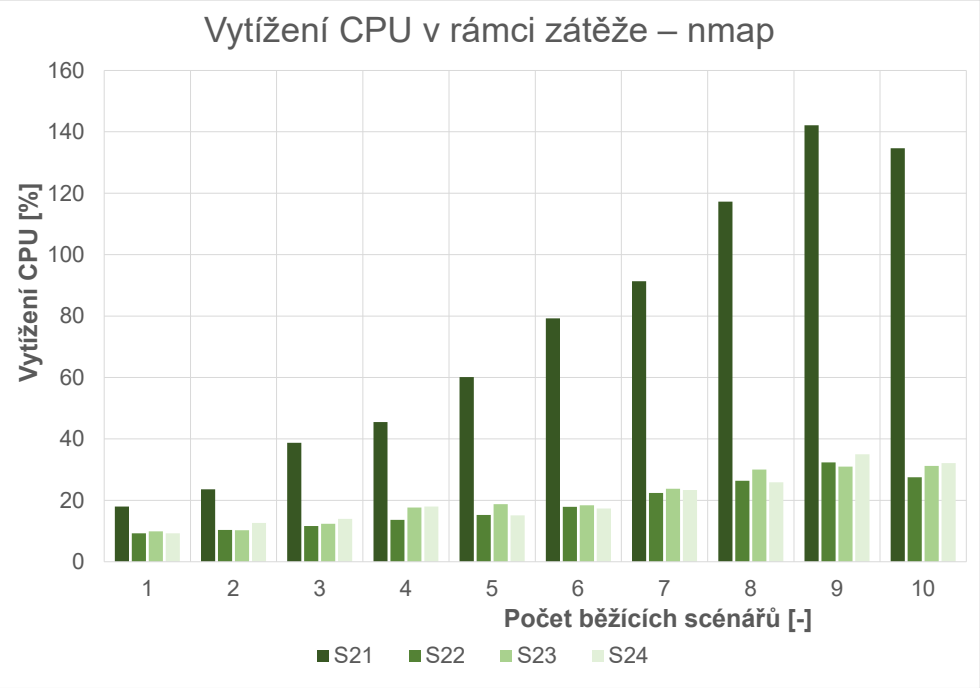
Zátěž – nmap



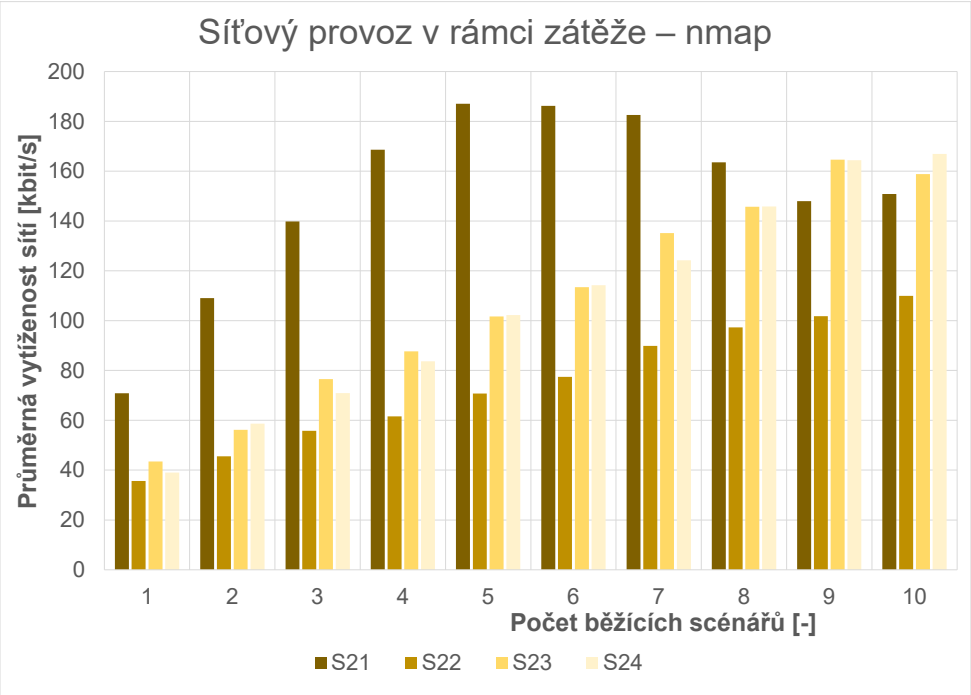
Obr. 3.15: Graf využití RAM pro scénáře S21–S24 během zátěže `nmap`



Obr. 3.16: Graf využití disku pro scénáře S21–S24 během zátěže `nmap`

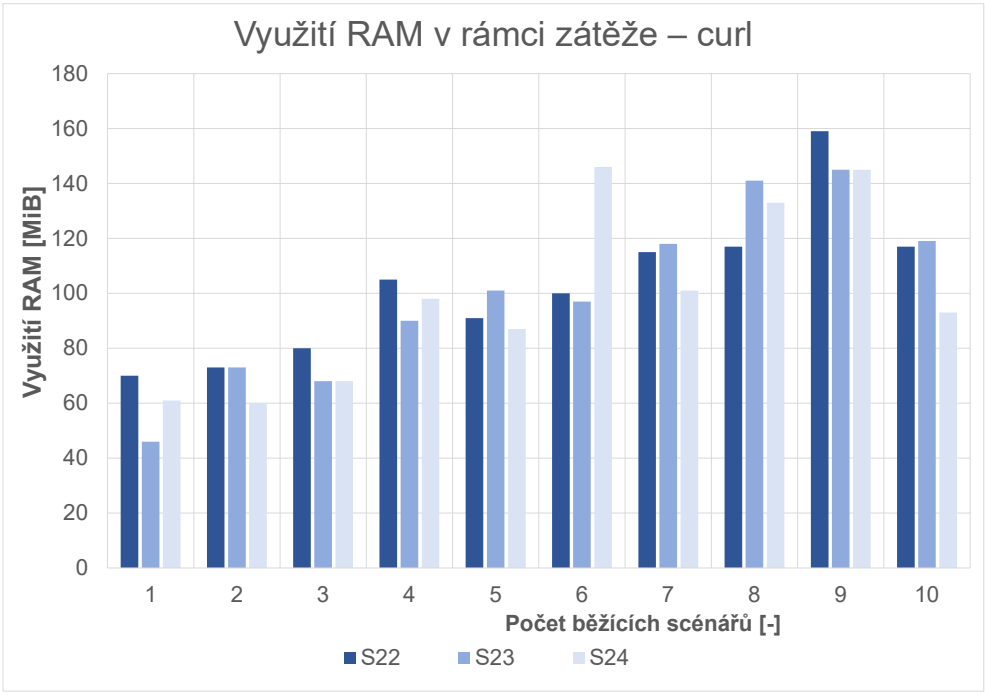


Obr. 3.17: Graf celkového vytížení CPU pro scénáře S21–S24 během zátěže `nmap`

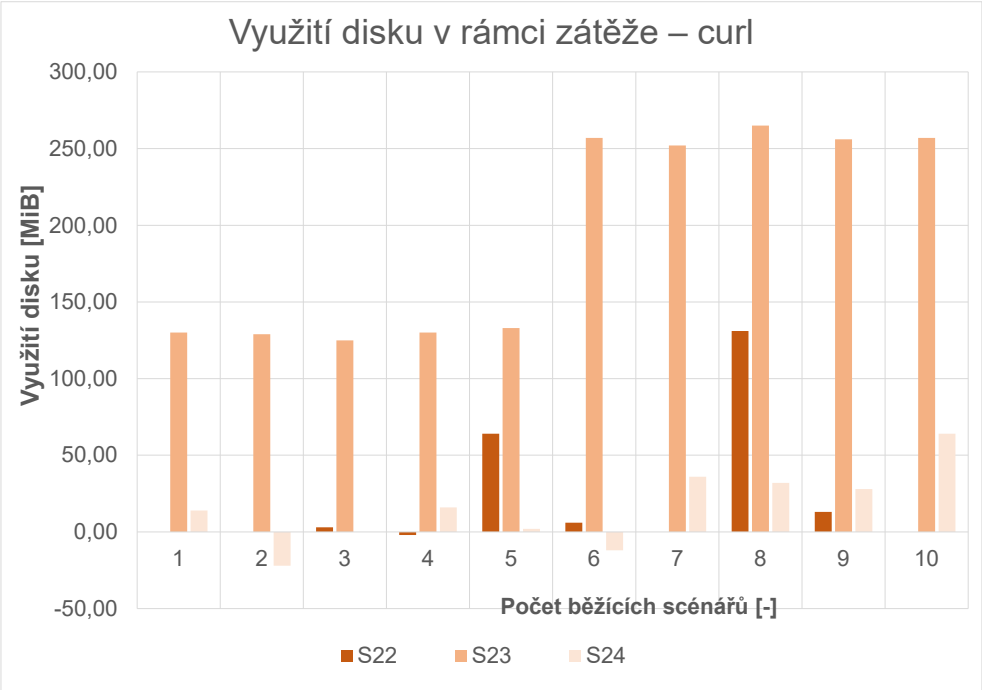


Obr. 3.18: Graf prům. vytížení sítí v rámci scénářů S21–S24 během zátěže `nmap`

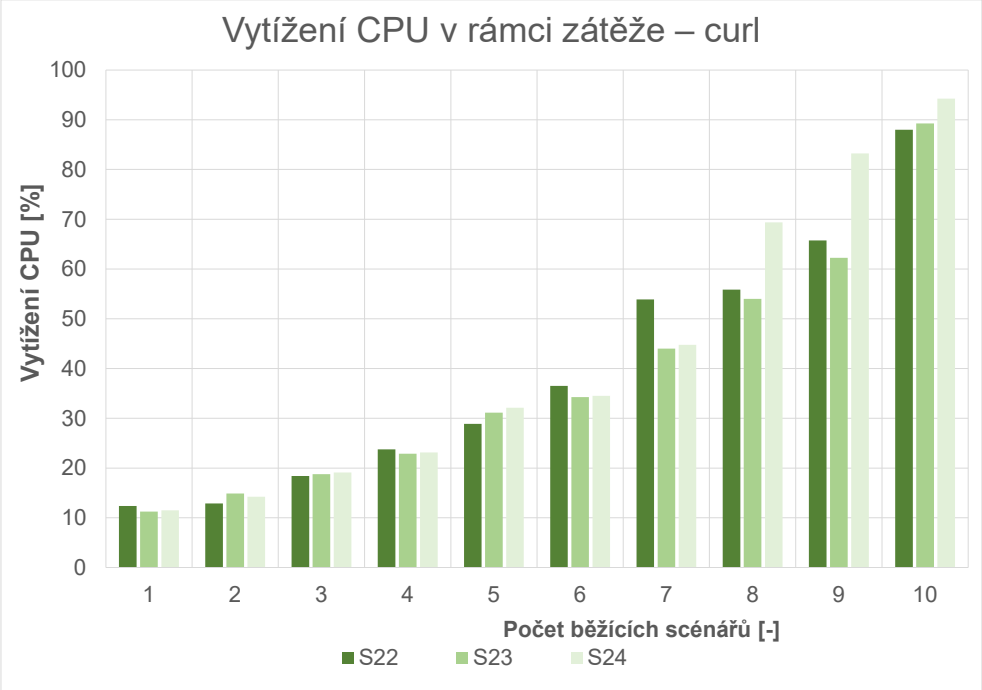
Zátěž – curl



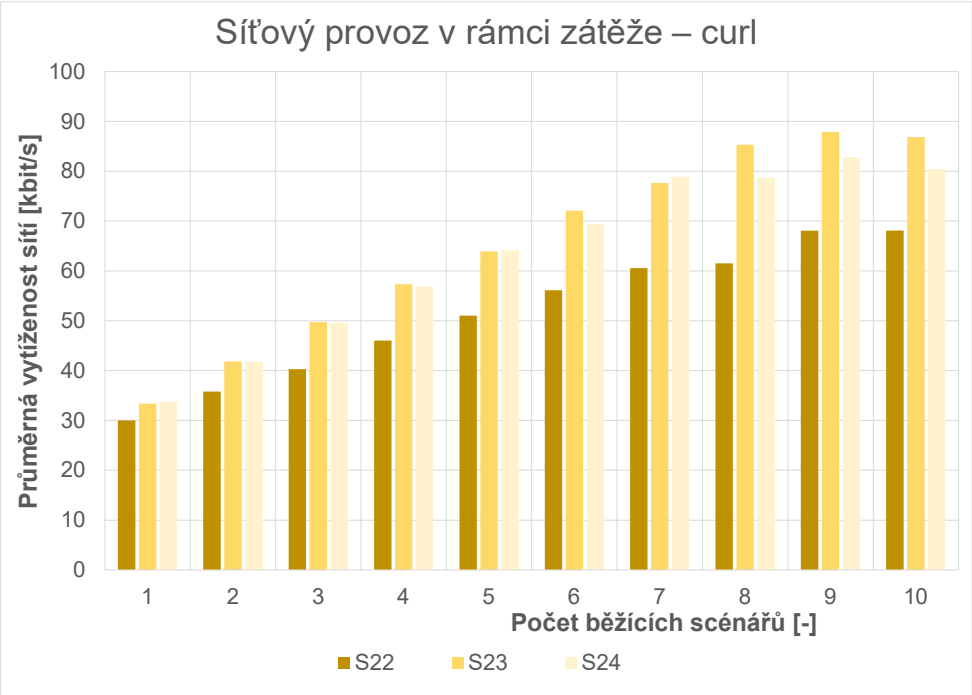
Obr. 3.19: Graf využití RAM pro scénáře S22–S24 během zátěže `curl`



Obr. 3.20: Graf využití disku pro scénáře S22–S24 během zátěže `curl`

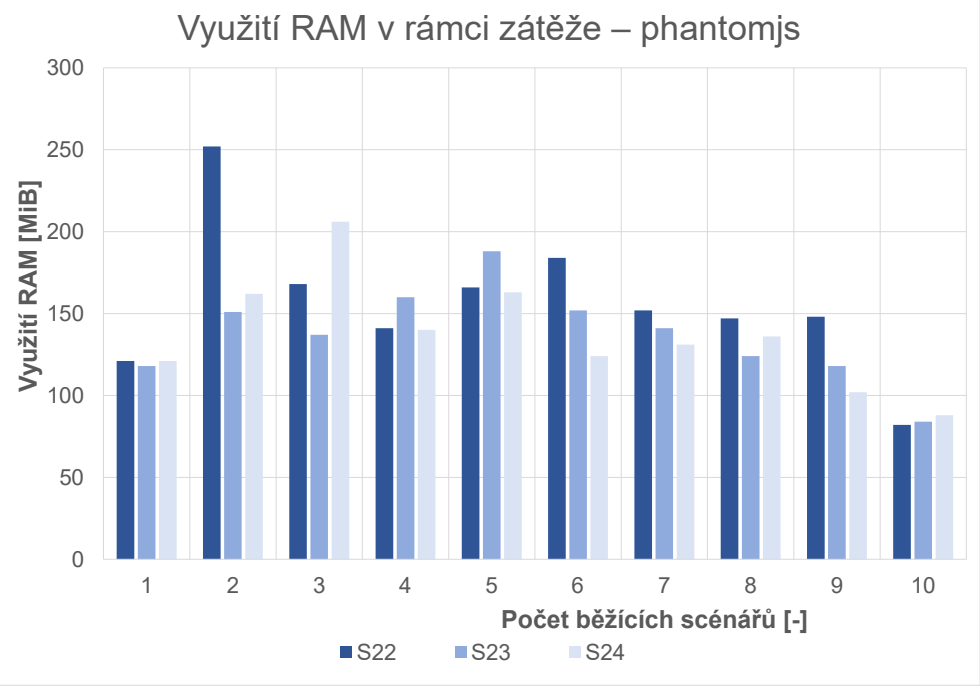


Obr. 3.21: Graf celkového vytížení CPU pro scénáře S22–S24 během zátěže `curl`

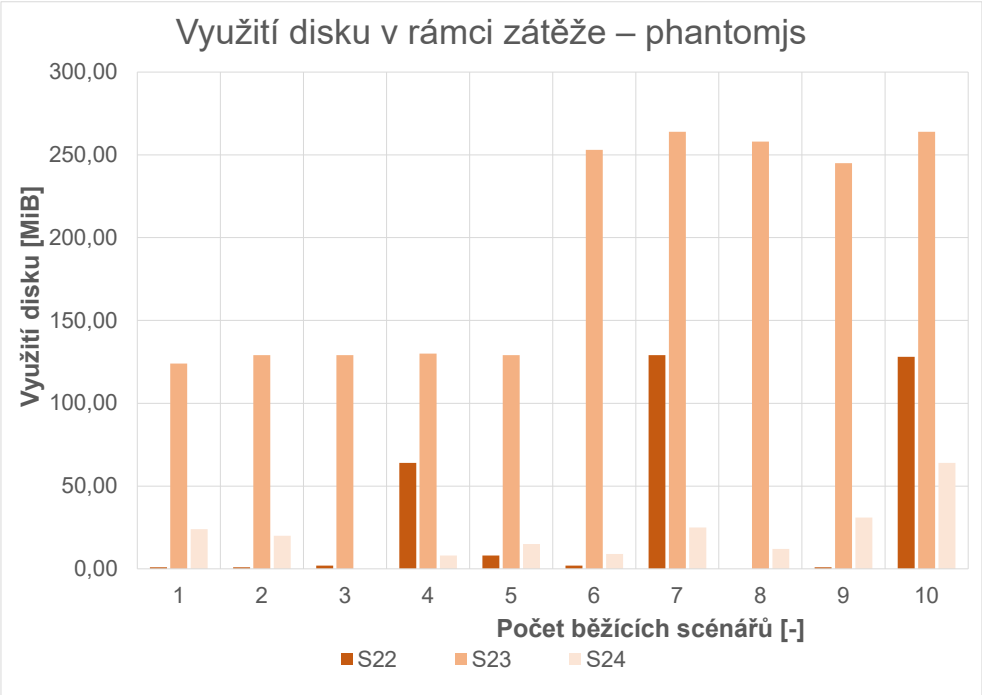


Obr. 3.22: Graf prům. vytížení sítí v rámci scénářů S22–S24 během zátěže `curl`

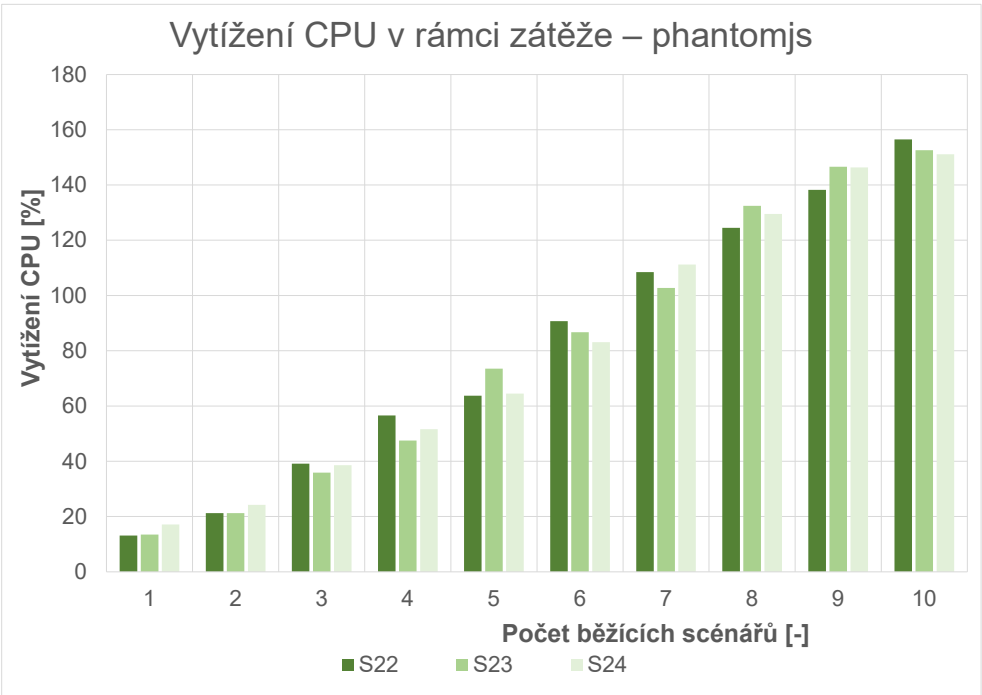
Zátěž – phantomjs



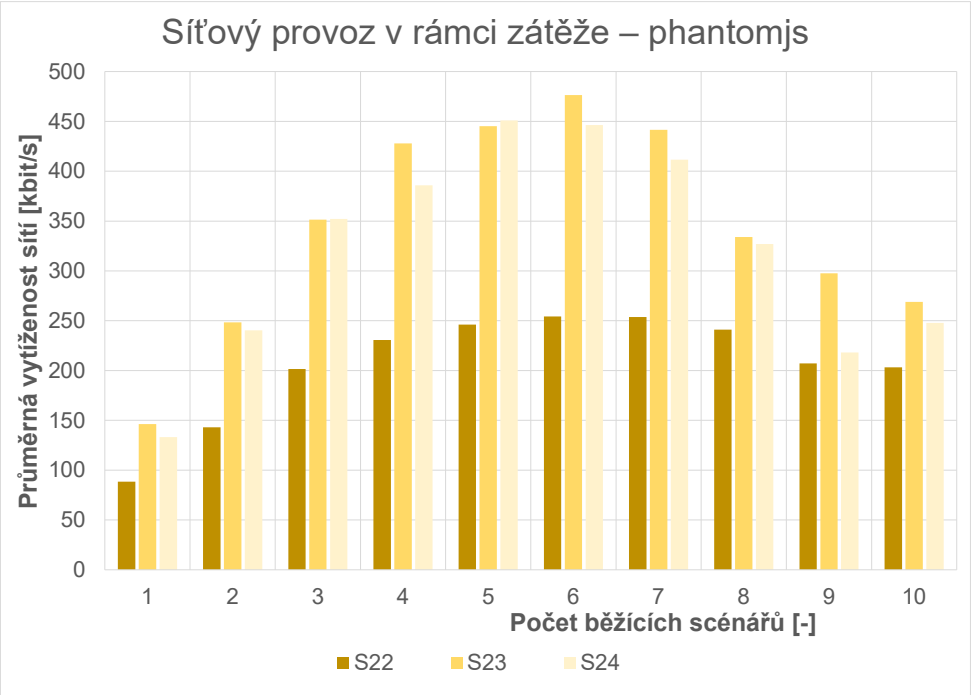
Obr. 3.23: Graf využití RAM pro scénáře S22–S24 během zátěže `phantomjs`



Obr. 3.24: Graf využití disku pro scénáře S22–S24 během zátěže `phantomjs`



Obr. 3.25: Graf celkového vytížení CPU pro scénáře S22–S24 během zátěže `phantomjs`

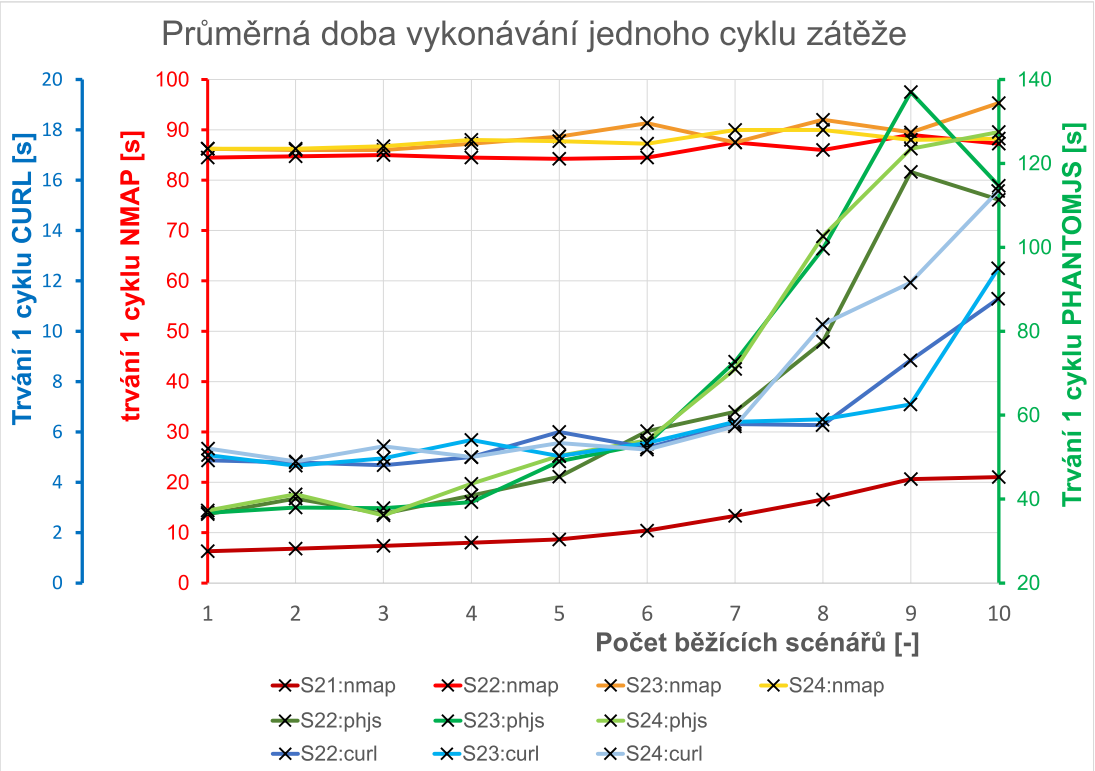


Obr. 3.26: Graf prům. vytížení sítí v rámci scénářů S22–S24 během zátěže `phantomjs`

3.4 Shrnutí výsledků měření

Porovnáním grafů s využitím RAM lze usoudit, že právě vytváření virtuální infrastruktury má na něj nejvyšší vliv, viz např. grafy 3.12 a 3.15. OpenStack při dosažení velikosti logovacích souborů (zhruba 400–520 MiB) archivuje příliš velké logy. V měření byla tato činnost mnohokrát zaznamenána (např. záporné hodnoty využití disku v grafech 3.16 a 3.20). V budoucnu tak v tomto směru nehrozí zbytečné využívání pevného disku. Naopak na výpočetní síle CPU jsou nejvíce závislé činnosti kontejnerů. Pokud je systém přetížen, obsluha procesů je zpomalena, viz graf 3.27 znázorňující průměrnou dobu vykonávání jednoho cyklu zátěže s rozdělením do `nmap` (červeno-oranžová), `curl` (modrá) a `phantomjs` (zelená). Ve výsledku je snížen i průtok dat napříč virtuálními sítěmi a nebo generování datového toku (např. S21 v grafu 3.18 nebo graf 3.26). Ze třetí skupiny měření dále vyplývá:

- První nasazení scénáře má o hodně vyšší využití RAM než následující, jelikož i služby OpenStacku potřebují RAM ke svým procesům (po restartování Dockeru jsou všechny zdroje využívané OpenStackem uvolněny; grafy 3.11 a 3.12).
- Oproti kontejneru útočníka Kali je server WebGoat závislý na diskovém prostoru (graf 3.13). Server WebGoat ke své činnosti spotřebovává i mnohem více RAM (rozdíl mezi S21 a S22 z grafu 3.12).



Obr. 3.27: Graf průměrného vykonávání 1 cyklu zátěže dle počtu běžících scénářů

- Pokud je útočník a server každý v jiné síti a komunikují přes směrovač (S23), scénář nemá na HW zdroje výrazný vliv v porovnání s S22, kdy jsou oba ve stejné síti (nepatrná zvýšená náročnost na RAM je zaznamenána jen u zátěže s `nmap`, viz graf 3.15).
- Komunikace přes firewall (S24) taktéž nemá na HW zdroje moc následky v porovnáním se směrovačem (S23), i když lze na grafu 3.21 vidět u počtu běžících scénářů 8–10 vyšší zátěž pro S24. U zátěže `nmap` a `phantomjs` je již vytížení CPU na takřka stejné úrovni.
- Výrazné využití disku S23 ve všech třech zátěžích vzniklo z doposud neznámých příčin. Navíc využití disku bylo vždy jen po dobu spuštěné zátěže. Po zastavení zátěže se velikost volného disku vrátila do podobného stavu jako před měřením.
- Vyšší vytížení ve scénáři S21 u zátěže `nmap` je způsobeno vyšší četností skenování, jelikož v síti není žádný web-server a 1 cyklus zátěže pak trvá 10krát méně oproti S22–S24. Z toho taktéž vyplývá, že samotné skenování sítě je náročnější než skenování portů serveru.
- Vyšší datový průtok u scénářů S23 a S24 je způsoben dvojnásobně vyšším počtem rozhraní mezi komunikujícími kontejnery (grafy 3.18, 3.22 a 3.26).

Na konci měření všech scénářů byla zkontrolována ztrátovost paketů. Celkově byl nalezen pouze 1 zahozený paket u zatěžování scénáře S24. Z pohledu celkového

provozu je ale tato ztrátovost tak minimální, že ji nelze spojovat s funkcionalitou OpenStacku.

Nakonec je změřeno používání Horizonu: prvotní načtení úvodní stránky spotřebuje okolo 1248 MiB paměti RAM a po načtení všech hlavních stránek z navigačního panelu stouplo využití o dalších 685 MiB. Při využívání Horizonu by tedy měl kontrolér disponovat minimálně o 2 GiB RAM navíc.

Závěr

Hlavním cílem práce bylo prozkoumat možnosti platforem s plnou podporou virtualizace a budování komunikační infrastruktury. V první části byly vysvětleny základní technologie jako virtualizace, cloud computing, kontejnerizace a jejich zástupci: KVM-QEMU pro virtualizaci a Docker pro kontejnerizaci. Pro zástupce cloud computingu byly následně podrobněji rozebrány platformy: Kubernetes, OpenStack a OpenShift.

Z hlediska stanovených požadavků pro vytváření virtuální infrastruktury byly u každé platformy rozebrány její hlavní vlastnosti a architektura s vysvětlením základních komponent. Dále kontejnerizace (včetně možností sandboxingu), síťování, tvorba firewallů a možnosti instalace s minimálními HW požadavky pro jejich nasazení. V Kubernetes byla vysvětlena metodika při vytváření kontejnerů a detailní způsoby síťování pro komunikaci mezi kontejnery v rámci stejného podu, mezi pody a Kubernetes a nakonec mezi pody samotnými. Vlastnosti OpenStacku vychází z jeho zvolených služeb. Pro kontejnerizaci byly popsány možnosti se službami Zun a Magnum. Síťování a firewally plně zajišťuje služba Neutron, která spolu se službou Kuryr poskytuje i propojení s Dockerem. OpenShift využívá všech komponent Kubernetes a navíc doplňuje chybějící vlastnosti, nebo vylepšuje jeho stávající.

Na základě popsaných a prozkoumaných možností byly shrnuty výhody a nevýhody všech tří platforem. Jako nejvhodnější byla vybrána platforma OpenStack, jelikož splňuje všechny kritéria zadané vedoucím této práce (podrobněji v kap. 1.7.1).

Úvod praktické části byl věnován přípravě prostředí a popisu parametrů vytvořeného VM. Následně bylo mnohokrát prováděno nasazení platformy OpenStack na VM a to až do chvíle, dokud nebyly všechny služby plně funkční. Nalezené chyby byly způsobené většinou špatnou konfigurací (např. rekonstrukce obrazů kontejnerových služeb Zun a Kuryr pomocí balíčkovacího systému – namísto ze zdrojových kódů; nesprávné nastavení Etcd uložiště, atd.), nebo špatně zvolenými službami (např. absence pluginu Etcd pro kontejnerové služby), které bylo možné opravit pouze opětovným nasazením. Pro splnění všech požadavků byly nakonec pro nasazení OpenStacku vybrány služby: Keystone, Nova, Glance, Placement, Neutron, Neutron-FWaaS, Horizon, Cinder, Cinder-backend-lvm, Zun, Kuryr a plugin Etcd. Následovalo hlubší testování funkčnosti jednotlivých služeb pomocí OpenStack klienta.

Nasazení a nastavení OpenStacku může být pro nové uživatele velmi matoucí. Právě z tohoto důvodu byl vypracován samostatný manuál k OpenStacku, který by měl většinu nejasností zodpovědět. Ze začátku je rozvedena příprava prostředí; dále konfigurace a instalace Openstacku, prvotní nastavení administrátorem a přidání nebo odebrání dalšího serveru do cloudu OpenStacku. Druhá kapitola manuálu

vysvětluje obecný postup při řešení problémů, nalezené problémy během a po nasazení OpenStacku, jakým způsobem je ověřována správná funkčnost všech služeb a v poslední řadě problémy, na které autor práce narazil během testování funkčnosti OpenStacku. Například na jeden nalezený problém byl založen bug report 1859176 – „nemožnost“ specifikovat běhové prostředí `kata-runtime` při vytváření kontejneru službou Zun. Nahlášený problém byl vyřešen a vývojáři již opraven ve verzích OpenStacku Stein a výš. Poslední dvě kapitoly manuálu přestavují uživatelské příručky pro práci v příkazovém řádku nebo přes grafické rozhraní Horizonu. Manuál k OpenStacku, který je taky vlastním přínosem práce, byl po konzultaci s vedoucím práce odevzdán mimo hlavní práci a to z důvodu velkého rozsahu dalších 44 stran.

Poslední část práce se zabývá samotným měřením náročnosti vytvořených scénářů v OpenStacku. Měření bylo rozděleno do tří skupin scénářů. První skupina se skládá ze 14 malých scénářů, které se zaměřují na náročnost dílčích příkazů OpenStacku. Ve druhé skupině byl implementován pokročilejší scénář (navíc se simulací zátěže), ze kterého lze vyvodit chování OpenStacku i přes značné přetížení systému. Jelikož z předchozích měření nebylo možné jednoznačně určit, jaký vliv mají na HW zdroje jednotlivé scénáře při zátěži, byly vytvořeny další 4 komplexní scénáře S21–S24 (třetí skupina). Základní scénář S21 vytváří sandbox útočníka v privátní síti s jedním směrovačem; S22 přidává do sítě web-server; S23 – každý sandbox je připojený do vlastní sítě; S24 přenastavuje směrovač na firewall.

V rámci testování náročnosti bylo vytvořeno celkem 41 skriptů převážně v jazyce Bash pro: skenování vytíženosti systému (3); vytváření a mazání scénářů (19); simulaci zátěže (5+1); automatizaci (9); a transformaci surových dat do čitelného/použitelného stavu (4). U pokročilejších scénářů byly vytvořeny i vlastní kontejnerové obrazy (4) přizpůsobené potřebám testování. Výstupy měření byly přepracovány do grafické podoby na konci každé skupiny měření.

Na základě prezentovaných výsledků měření lze usoudit, že budování virtuální infrastruktury má největší vliv na využití RAM. Menší využití disku je způsobeno logováním OpenStacku a zápisem do SQL databáze MariaDB. Využití disku během měření i mnohokrát poskočilo do záporných čísel, to je způsobeno archivací příliš velkých logovacích souborů, která se prováděla automaticky OpenStackem (při velikosti složky s logy 400–520 MiB). Samotný běh a práce v kontejnerech jsou dle očekávání nejvíce závislé na výkonu procesoru výpočetních uzlů. V rámci simulací zátěže scénářů S15 a S21–S24 docházelo velmi často k přetížení CPU u vyššího počtu opakování běžících scénářů. Nejvíce byl přetížen scénář S15 spuštěním zátěže `s16` a to až na 2,8násobek. Během měření ale nebylo zjištěno, že by tak vysoké přetížení výrazně ovlivňovalo OpenStack na funkčnosti. Na síťových rozhraních nebyly nalezeny žádné hromadné chyby, kolize nebo zahazování paketů (1 zahozený paket se objevil během zátěže S24, z pohledu celkového provozu je ale tato ztráta

tak minimální, že ji s funkcionalitou OpenStacku nezle spojovat). Přetížení procesoru vedlo hlavně k pomalejším kontejnerům a snížení datového provozu v sítích (např. u S15 se provoz snížil na 42,1 % předchozí velikosti). Dále bylo zjištěno, že umístění útočníka a web-serveru do jiných sítí (S23), nemá výrazný vliv na využití HW zdrojů oproti umístění obou do stejné sítě (S22). Stejně tak rozdíl propojení firewallem (S24), namísto směrovače (S23), nemá na HW zdroje významné následky. Podrobnější výsledky měření jsou shrnuty v kapitole 3.4.

Tímto byly veškeré stanovené cíle práce splněny. V budoucnu by zpracování náročnosti scénářů mohlo být vypracováno i v závislosti na čase. Experimentálně byl přepsán skript `s31` a skript měření `s00`, který nyní snímá vytížení systému každých 10 s. Oběma skriptům byly přidány časové značky, aby je bylo možné vyhodnotit dohromady. Takto byl přeměřen scénář S21 (skripty a výsledky dostupné v e-příloze), výstup ale dále nebyl zpracován do grafické podoby. Pokud by data byly porovnány s aktuálně naměřenými, nepřineslo by to pravděpodobně skoro žádné nové zjištění a pokud by byly naměřeny i další scénáře, výstup by byl velmi pravděpodobně nekonzistentní (vzhledem k době odevzdání práce). Celkově by se hodnoty od naměřených výrazně nezměnily, ale z výstupu by bylo navíc možné zjistit, jak je náročný samotný průběh vytváření a mazání scénářů na procesor a stejně tak na alokaci a uvolňování paměti RAM.

Pokračováním této diplomové práce bude nasazení OpenStacku na v budoucnu zakoupené servery i se vším nastavením, potřebným k jeho správné funkčnosti. A dále pravděpodobně vytváření a spouštění scénářů pro studenty studijního programu Informační bezpečnosti.

Literatura

[1] HUANG, D.; WU, H. *Mobile cloud computing: Foundations and service models.* Cambridge, MA, United States: Morgan Kaufmann Publishers, an imprint of Elsevier, c2018. 336 s. ISBN 978-0-12-809641-3.

[2] *Understanding virtualization.* RedHat [online]. c2019 [cit. 20. 10. 2019]. Dostupné z WWW: <https://www.redhat.com/en/topics/virtualization>.

[3] CHIRAMMAL, H. D.; MUKHEDKAR, P.; VETTATHU, A. *Mastering KVM Virtualization.* UK Birmingham: Packt Publishing Ltd., c2016. 468 s. ISBN 978-1-78439-905-4.

[4] *What does Cloud Orchestration mean? – definition from Techopedia.* [online]. c2019 [cit. 20. 10. 2019]. Dostupné z WWW: <https://www.techopedia.com/definition/32988/cloud-orchestration>.

[5] *What is a Container?* Docker [online]. c2019 [cit. 3. 11. 2019]. Dostupné z WWW: <https://www.docker.com/resources/what-container>.

[6] INSU, J. *Container Runtime* Better Tomorrow with Computer Science [online]. c2017 [cit. 31. 10. 2019]. Dostupné z WWW: <https://insujang.github.io/2019-10-31/container-runtime/>.

[7] TRACEY, C.; BURNS, B. *Managing Kubernetes: operating Kubernetes clusters in the real world.* 2nd. Sebastopol, CA: O'Reilly Media, 2018. 187 s. ISBN 978-1-492-03391-2.

[8] *Docker Documentation.* Docker docs [online]. c2019 [cit. 3. 11. 2019]. Dostupné z WWW: <https://docs.docker.com/>.

[9] Bayern, M. *The 10 most popular container tools for businesses.* TechRepublic [online]. c2019 [cit. 27. 2. 2019]. Dostupné z WWW: <https://www.techrepublic.com/article/the-10-most-popular-container-tools-for-businesses/>.

[10] *What is Docker?* RedHat [online]. c2019 [cit. 3. 11. 2019]. Dostupné z WWW: <https://www.redhat.com/en/topics/containers/what-is-docker>.

[11] *Concepts.* Kubernetes [online]. c2019 [cit. 3. 11. 2019]. Dostupné z WWW: <https://kubernetes.io/docs/concepts/>.

[12] *CNCF – Project type: Kubernetes; Commits by Company.* Stackalytics [online]. [cit. 19. 5. 2020]. Dostupné z WWW: <https://www.stackalytics.com/cncf?project_type=kubernetes>.

[13] Caban, W. *Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and Operations Teams.* Columbia, MD: Apress, 2019. ISBN 978-1-4842-4984-0.

[14] LUKSA, M. *Kubernetes in action.* Shelter Island, New York: Manning Publications, 2017. ISBN 978-161-7293-726.

[15] The gVisor Authors. *What is gVisor?* gVisor [online]. c2020 [cit. 18. 5. 2020]. Dostupné z WWW: <https://gvisor.dev/docs/>.

[16] TAMURA, Y.; SCANNEL, A. *GKE Sandbox: Bring defense in depth to your pods.* Google Cloud [online]. c2020 [cit. 15. 5. 2019]. Dostupné z WWW: <https://cloud.google.com/blog/products/containers-kubernetes/gke-sandbox-bring-defense-in-depth-to-your-pods>.

[17] *Understanding VXLANs.* Juniper Networks [online]. c1999–2019 [cit. 11. 7. 2019]. Dostupné z WWW: <https://www.juniper.net/documentation/en_US/junos/topics/topic-map/sdn-vxlan.html>.

[18] REDDY, M. *Kube-router.* [online]. [cit. 4. 12. 2019]. Dostupné z WWW: <https://www.kube-router.io/>.

[19] *Introduction – Networking, Security.* Project Calico [online]. [cit. 4. 12. 2019]. Dostupné z WWW: <https://docs.projectcalico.org/v3.10/introduction/>.

[20] *Getting Started – Features, Policies.* Contiv [online]. c2017 [cit. 4. 12. 2019]. Dostupné z WWW: <https://contiv.io/documents/gettingStarted/>.

[21] *Introducing Weave Net.* Weaveworks [online]. 2014–2019 [cit. 4. 12. 2019]. Dostupné z WWW: <https://www.weave.works/docs/net/latest/overview/>.

[22] *Getting started.* Kubernetes [online]. c2019 [cit. 5. 12. 2019]. Dostupné z WWW: <https://kubernetes.io/docs/setup/>.

[23] *Ansible Documentation.* Ansible [online]. c2019 [cit. 4. 12. 2019]. Dostupné z WWW: <https://docs.ansible.com/ansible/latest/index.html>.

[24] *Overview, Openstack Components, SDKs, Deployment Tools.* OpenStack [online]. [cit. 7. 12. 2019]. Dostupné z WWW: <https://www.openstack.org/software/>.

[25] BUMGARDNER, V. K. Cody. *OpenStack in action.* Shelter Island, New York: Manning Publications., 2016. ISBN 978-161-7292-163.

[26] *OpenStack Foundation – Project type: OpenStack; Commits by Company.* Stackalytics [online]. [cit. 19. 5. 2020]. Dostupné z WWW: <https://www.stackalytics.com/?period=grizzly&metric=commits&project_type=all&release=all>.

[27] *OpenStack Installation Guide.* OpenStack docs [online]. [cit. 2. 12. 2019]. Dostupné z WWW: <https://docs.openstack.org/install-guide/index.html>.

[28] KUMARI, M.; SINGH, P. K. *Containers in OpenStack.* Shelter Island, NY: Packt Publishing, 2017. ISBN 9781788394383.

[29] NOVA SUBTEAM. *Docker.* OpenStack wiki [online]. [cit. 17. 10. 2016]. Dostupné z WWW: <https://wiki.openstack.org/wiki/Docker>.

[30] *Welcome to Zun's documentation!.* OpenStack docs [online]. [cit. 22. 7. 2019]. Dostupné z WWW: <https://docs.openstack.org/zun/latest/>.

[31] *Welcome to Magnum's Developer Documentation!.* OpenStack docs [online]. [cit. 23. 4. 2019]. Dostupné z WWW: <https://docs.openstack.org/magnum/latest/>.

[32] *The speed of containers, the security of VMs.* katacontainers [online]. [cit. 18. 5. 2020]. Dostupné z WWW: <https://katacontainers.io/>.

[33] *OpenStack Security Guide.* OpenStack docs [online]. [cit. 2. 12. 2019]. Dostupné z WWW: <https://docs.openstack.org/security-guide/index.html>.

[34] *Welcome to Neutron's documentation!.* OpenStack docs [online]. [cit. 13. 7. 2019]. Dostupné z WWW: <https://docs.openstack.org/neutron/latest/index.html>.

[35] NI, Z. *Adding Fuxi as a Kuryr sub-project.* OpenDev [online]. [cit. 2. 8. 2016]. Dostupné z WWW: <https://review.opendev.org/#/c/347083/>.

[36] *DevStack.* OpenStack docs [online]. [cit. 25. 9. 2019]. Dostupné z WWW: <https://docs.openstack.org/devstack/latest/>.

[37] *Kolla-Ansible Deployment Guide.* OpenStack docs [online]. [cit. 30. 9. 2019]. Dostupné z WWW: <https://docs.openstack.org/project-deploy-guide/kolla-ansible/latest/index.html>.

[38] *Red Hat OpenShift – Products* RedHat [online]. c2019 [cit. 4. 12. 2019]. Dostupné z WWW: <https://www.redhat.com/en/technologies/cloud-computing/openshift#?>.

[39] *Other Projects – Project type: OpenShift; Commits by Company.* Stackalytics [online]. [cit. 19. 5. 2020]. Dostupné z WWW: <https://www.stackalytics.com/unaffiliated?project_type=openshift-group&release=all&metric=commits>.

[40] USOV, A.; KROPACHEV, A.; ZUEV, D. *Learn OpenShift.* UK Birmingham: Packt Publishing Ltd., c2018. ISBN 978-1-78899-232-9.

[41] *OKD Documentation – Version: OKD 3.11.* okd docs [online]. [cit. 4. 12. 2019]. Dostupné z WWW: <https://docs.okd.io/>.

[42] *Continuous Integration (CI)* Techopedia [online]. c2019 [cit. 16. 12. 2019]. Dostupné z WWW: <https://www.techopedia.com/definition/24368/continuous-integration-ci>.

[43] *Continuous Delivery (CD)* Techopedia [online]. c2019 [cit. 16. 12. 2019]. Dostupné z WWW: <https://www.techopedia.com/definition/28958/continuous-delivery-cd>.

[44] *[Dual CPU] Intel Xeon E5-2630L v2 - CPU benchmark, performance score* CPU benchmarks [online]. [cit. 20. 5. 2020]. Dostupné z WWW: <https://cpu-benchmarks.com/cpu/dual-cpu-intel-xeon-e5-2630l-v2-2-40ghz/>.

Seznam zkratek

ACL	Access Control List
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
App	Application
BGP	Border Gateway Protocol
BUM	Broadcast, Unkown destination address, Multicast
CCM	Cloud Controller Manager
CD	Continuous Delivery, Continuous Development
CI	Continuous Integration
CLI	Command Line
CNI	Container Network Interface
COE	Container Orchestration Engine
CPU	Central Processing Unit
CRI	Container Runtime Interface
CRUD	Create, Read, Update, Delete
DHCP	Dynamic Host Configuration Protocol
DNAT	Destination Network Address Translation
DNS	Domain Name System
DVR	Distributed Virtual Router
FTP	File Transfer Protocol
FWaaS	Firewall as a Service
GKE	Google Kubernetes Engine
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HA	High Availability
HDD	Hard Disk Drive
HTTP	The Hypertext Transfer Protocol
HW	Hardware
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPAM	IP Address Manager
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
K8s	Kubernetes
KCM	Kube Controller Manager
Kind	Kubernetes IN Docker
KVM	Kernel-based Virtual Machine

LDAP	Lightweight Directory Access Protocol
LVM	Logical Volume Management
ML2	Modular Layer 2
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIC	Network Interface Card
NTP	Network Time Protocol
OCI	Open Container Initiative
OKD	Origin Community Distribution
OS	Operating System
OSF	OpenStack Foundation
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OVS	OpenVSwitch
QEMU	Quick EMUlator
QoS	Quality of Services
RAM	Random Access Memory
RBAC	Role-Based Access Control
RHEL	Red Hat Enterprise Linux
RX	Receive
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Network
SNAT	Source Network Address Translation
SQL	Structured Query Language
SR-IOV	Single Root I/O Virtualization
SSD	Solid-State Drive
SSH	Secure Shell
SW	Software
S*xx*	Scenario with number *xx*
s*xx*	Script with number *xx*
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TX	Transmit
UDP	User Datagram Protocol
VIP	Virtual IP
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNID	Virtual Network ID
VPNaaS	Virtual Private Network as a Service

VXLAN Virtual Extensible Local Area Network
YAML Ain't Markup Language

Seznam příloh

A Obsah přiloženého datového média

Přiložené datové médium obsahuje elektronickou verzi diplomové práce.

- `/` — Kořenový adresář datového média
  - `DP_Virtualizace-infrastruktury_Stodulka.pdf` — Hlavní dokument práce
  - `Docker_images_creation` — Vše pro vytvoření kontejnerových obrazů
    - `FTP-server`
      - `names.txt`
      - `readme.txt` — Postup pro vytvoření FTP serveru včetně jeho konfigurace
    - `HTTP-FTP-client`
      - `Dockerfile`
      - `ftp.sh` — Skript simulující zátěž stahováním souboru z FTP serveru
      - `http.sh` — Skript simulující zátěž stahováním webové stránky
    - `HTTP-server`
      - `Dockerfile`
      - `index.html`
    - `Kali-linux`
      - `00save_info` — Menší databáze kontejneru s čítačem spuštění skriptů
      - `Dockerfile`
      - `phscript.js` — Skript pro `phantomjs` v jazyce javascript
      - `script.sh` — Skript pro simulaci zátěže: `nmap`, `curl` a `phantomjs`
      - `vimrc_setting` — Přizpůsobení vimu pro práci v kontejneru
  - `Measured_data`
    - `raw-data` — Výstupy ze skriptů 17, 31–38 a trvání cyklů zátěže z 31–34
      - `...`
    - `data-output.xlsx` — Zpracované výstupy ze všech měření včetně grafů
  - `Scripts`
    - `time-dependent` — Skripty pro měření v časové závislosti, včetně dat
      - `00-measurement.sh4`
      - `51-execute-S21-timeline`
      - `...`
    - `00-measurement.sh` — 00–00v3: skripty měření
    - `00-measurement.sh2`
    - `00-measurement.sh3`
    - `01-network.sh` — Skripty 01–14: **první** řada scénářů
    - `02-router.sh`
    - `03-attach-to-router.sh`
    - `04-firewall-rule.sh`
    - `05-firewall-policy.sh`
    - `06-firewall-policy-add-rule.sh`
    - `07-firewall-group-create.sh`
    - `08-firewall-group-add-ingress-policy.sh`
    - `09-attach-firewall-to-router.sh`
    - `10-create-project.sh`
    - `11-create-user-into-project.sh`

- `12-create-vm.sh`
- `13-create-container.sh`
- `14-create-sandbox-container.sh`
- `15-scenario1-setup.sh` ... Skripty 15–17: **druhá** řada scénářů
- `16-scenario1-workload.sh` ... Spouštěč zátěže
- `17-scenario1-execute-normal.sh` .. Automatizace 00v2 a 15, pro RAM/disk
- `20-new-scenario-workload.sh` ... Skripty 20–38: **třetí** řada scénářů, 20: spouštěč zátěže
- `21-new-scenario1-setup.sh` ... 21–24: vytváření scénářů
- `22-new-scenario2-setup.sh`
- `23-new-scenario3-setup.sh`
- `24-new-scenario4-setup.sh`
- `31-execute-1-1.sh` ... 31–34: automatizace skriptů 00, 20–24
- `32-execute-2-123.sh`
- `33-execute-3-123.sh`
- `34-execute-4-123.sh`
- `35-execute-normal.sh` ... 35–38: automatizace skriptů 00v2, 21–24, pro RAM/disk
- `36-execute-normal.sh`
- `37-execute-normal.sh`
- `38-execute-normal.sh`
- `40-values-printer.sh` ... 40–40v3: pro transformaci surových dat do čitelné podoby
- `40-values-printer2.sh`
- `40-values-printer3.sh`
- `41-avg-values.sh` ... Zprůměrování hodnot trvání na 1 cyklus zátěže

B Ukázka z Manuálu k OpenStacku

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

MANUÁL K OPENSTACKU
OPENSTACK MANUAL

PŘÍLOHA K DIPLOMOVÉ PRÁCI:
PLATFORMA PRO VIRTUALIZACI KOMUNIKAČNÍ INFRASTRUKTURY
MASTER'S THESIS ATTACHMENT:
COMMUNICATION INFRASTRUCTURE VIRTUALIZATION PLATFORM

BRNO 2020 Bc. Tomáš Stodůlka

Obsah

Úvod

Nasazení a nastavení OpenStacku může být pro nové uživatele velmi matoucí. Překážek – které je třeba do začátku překonat – je až příliš mnoho: ať už v orientaci mezi obrovským množství dokumentace, mnoha instalačních a konfiguracích možností, nebo v hledání řešení různých chybových stavů během instalace, či používání OpenStacku. Na základě toho je vytvořen tento manuál, který by měl většinu nejasností zodpovědět.

Instalační manuál uvažuje nasazení OpenStacku ve verzi Train pomocí nástroje `kolla-ansible` v režimu `all-in-one`, nebo-li vše v jednom virtuálním stroji (VM).

První kapitola **Instalace a nastavení** se zaměřuje na přípravu prostředí, konfiguraci a nasazení OpenStacku, jeho prvotní administrátorské nastavení (včetně sandboxingu pomocí kata kontejnerů) a přidání/odebrání dalšího OpenStack serveru. Druhá kapitola **Řešení problémů** popisuje obecný postup pro řešení chybového stavu, dále známé problémy během/po nasazení OpenStacku, problémy při jeho používání a taktéž manuál pro ověření správné funkčnosti všech služeb OpenStacku. Následující kapitoly **Uživatelská příručka – CLI** a **Uživatelská příručka – Horizon** popisující správu OpenStacku přes příkazový řádek nebo webový prohlížeč.

Použité konvence v průběhu dokumentace

```
Příkazový řádek, bash.
```

```
Editace textového souboru.
```

Blok s informačním/doplňujícím sdělením.

Blok s důležitým sdělením.

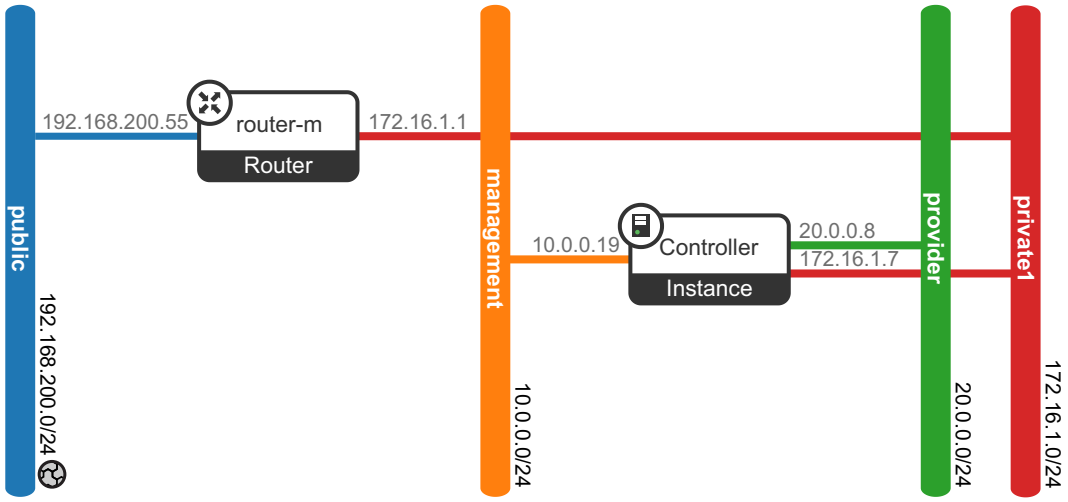
Blok označující chybovou hlášku.

Tab. 1: Konvence použité uvnitř textových bloků

Příklad	Popis konvence
<PROMĚNNÁ>	Proměnné jsou uváděny ve špičatých závorkách velkým písmem.
# komentář	Komentáře začínají hash symbolem.
"textový řetězec"	Textový řetězec je ohraničen uvozovkami s červenou barvou.
↪	Pokračování zalomeného řádku.
\	Zalomení příkazu.

1 Instalace a nastavení

Pro nasazení OpenStacku bude v tomto manuálu použita topologie dle obrázku 1.1 s jedním VM, jedním virtuálním směrovačem, jednou sítí a třemi virtuálními sítěmi. Síť `public` poskytuje připojení do Internetu. Směrovač `router-m` natuje provoz ze sítě `private` do sítě `public`. V případě režimu `multinode` je zapotřebí nastavit síťování mezi všemi uzly v ideálním případě alespoň s rychlostí 1 GB/s.



Obr. 1.1: Předpřipravená topologie

1.1 Příprava prostředí

Pokud je topologie umístěna za proxy, nastavte na všech uzlech **/etc/environment** jako například:

```
http_proxy=http://<IP:PORT>
https_proxy=http://<IP:PORT>
no_proxy=127.0.0.1,localhost,<X.X.X.1>,10.0.0.0/8,20.0.0.0/24  # kde X.X.X.1 je IP
  ↪adresa serveru na kterém běží VM s budoucím OpenStackem
```

Nastavte překlady IP adres na doménové jména:

```
127.0.0.1 localhost
::1 localhost
#10.0.0.19 Controller  ##multinode
#10.0.0.X Compute  ##multinode
```

Instalace závislostí

Povolte `epel` repositář a následně nainstalujte `pip` s aktualizací na jeho nejnovější verzi:

```
yum -y install epel-release
yum -y install python-pip
pip install -U pip
```

Nainstalujte závislosti použitím `yum` a ansible použitím `pip`:

```
yum -y install python-devel libffi-devel gcc openssl-devel libselinux-python
pip install ansible
```

Instalace Kolla a Kolla-ansible pro vývojáře

Nainstalujte `git` a následně naklonujte repositáře `kolla` a `kolla-ansible` do adresáře **root**:

```
yum -y install git
cd /root/
git clone -b stable/train https://github.com/openstack/kolla.git
git clone -b stable/train https://github.com/openstack/kolla-ansible.git
```

Nainstalujte požadavky pro kollu a kollu-ansible:

```
pip install -r kolla/requirements.txt --ignore-installed requests
pip install -r kolla-ansible/requirements.txt
pip install ./kolla  # od verze Train
pip install ./kolla-ansible  # od verze Train
```

Zkopírujte konfigurační soubory (**globals.yml**, **passwords.yml**) do **/etc/kolla**:

```
mkdir -p /etc/kolla
cp -r kolla-ansible/etc/kolla/* /etc/kolla
```

Zkopírujte soubory inventáře (**all-in-one**, **multinode**) do adresáře **root**:

```
cp kolla-ansible/ansible/inventory/* /root/
```

1.2 Konfigurace

Soubor ansible inventáře

Definujte soubor inventáře – **all-in-one** pro nasazení jednoho uzlu, nebo **multinode** pro 2 a více uzlů:

```
[control]
localhost       ansible_connection=local
#Controller  ##multinode
[network]
localhost       ansible_connection=local
#Controller  ##multinode
```