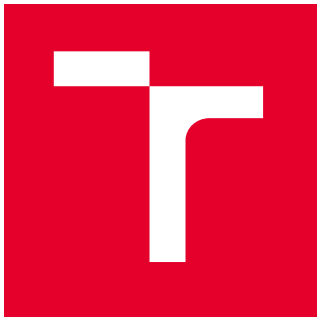


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

PŘÍPRAVA A IMPLEMENTACE JUMP SERVERU PRO HERNÍ SCÉNÁŘE V KYBERNETICKÉ ARÉNĚ

JUMP SERVER PREPARATION AND IMPLEMENTATION FOR GAME SCENARIOS IN CYBER ARENA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Ladislav Komárek**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Stodůlka**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Ladislav Komárek ***ID:*** 211798

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Příprava a implementace Jump serveru pro herní scénáře v Kybernetické aréně

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem práce je návrh a implementace rozhraní (Jump serveru) pro propojení virtuálních serverů ve scénářích Kybernetické arény s fyzickou sítí pro snadnou konektivitu pomocí SSH. Prostudujte a zpracujte problematiku týkající se virtualizace, kontejnerizace a propojování virtuální infrastruktury s fyzickou sítí. Vyberte a upravte diskový obraz, který bude co nejméně náročný na HW zdroje a bude splňovat všechny potřeby Jump serveru. Berte přitom v úvahu zda bude Jump server nasazen jako virtuální stroj nebo kontejner. Tento server bude především poskytovat SSH konektivitu z fyzické sítě přímo na virtuální stroje, které jsou součástí herních scénářů v Kybernetické aréně. Je potřeba brát na vědomí, že scénář je hratelný pro více studentů a můžou tak vznikat duplicity. Jump server by měl řešit tyto případné duplicity a tunelovat SSH spojení na dané virtuální stroje ve scénáři. Kybernetická aréna virtualizuje scénáře na cloudové platformě OpenStack pomocí orchestrační služby Heat. Cílem této práce bude zautomatizovat implementaci zmíněného Jump serveru do již vytvořené šablony Heatu, případně v Ansible.

DOPORUČENÁ LITERATURA:

[1] BARRETT, D. J.; SILVERMAN, R. E.; BYRNES, R. G. SSH, The Secure Shell: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.", 2005. ISBN 9780596008956

[2] LACROIX, J. Mastering Linux Network Administration. Packt Publishing Ltd, 2015. ISBN 978-1-78439-959-7

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: Ing. Tomáš Stodůlka

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se věnuje problematice ohledně kontejnerizace a virtualizace, zaměřuje se na platformu OpenStack a na vytváření Jump serveru. Hlavním cílem práce je vytvoření Jump serveru, který poskytuje SSH konektivitu k instancím v OpenStacku. Závěrečná práce je složena ze dvou částí – teoretické a praktické. V rámci teoretické části byla provedena analýza možností virtualizace a kontejnerizace pro Jump server. Součást teoretické části je samotný popis platformy OpenStack a jeho funkcí. Na základě teoretické analýzy byl vytvářen v praktické části Jump server. Po manuálním ověření funkčnosti Jump serveru bylo jeho vytváření automatizováno na platformu Openstack.

KLÍČOVÁ SLOVA

OpenStack, virtuální stroj, Jump server, virtualizace, kontejnerizace, Docker, automatizace

ABSTRACT

Bachelor's thesis deals with the problematics of the containerization and virtualization. Thesis focuses on the OpenStack platform and the creation of a Jump server. The main purpose of the thesis is to create a Jump server with the SSH connection to the OpenStack instances. Thesis is divided into two parts - theoretical and practical. The analysis of virtualization and containerization possibilities for Jump server is included in the theoretical part of the paper. Description of the OpenStack platform and its functions is also contained in the theoretical part. Based on the theoretical analysis, a Jump server was created in the practical part. After manually verifying the functionality of the Jump server, its creation was automated on the Openstack platform.

KEYWORDS

Openstack, virtual machine, Jump server, virtualization, containerization, Docker, automation

LADISLAV, Komárek. *Příprava a implementace Jump serveru pro herní scénáře v Kybernetické aréně*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 62 s. Bakalářská práce. Vedoucí práce: Ing. Tomáš Stodůlka

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Komárek Ladislav
VUT ID autora:	211798
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Příprava a implementace Jump serveru pro herní scénáře v Kybernetické aréně

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Tomáši Stodůlkovi za odborné vedení, pravidelné konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Vzhledem k digitalizace dat a využívaní moderních technologií, jenž využívají informační síť, se klade důraz na zabezpečení. S čímž souvisí vysoké nároky na informační bezpečnost. Proto je třeba podporovat vzdělání v tomto oboru. Kybernetická aréna je projekt, který má studentům přiblížit možné scénáře zranitelností, a také mohou získat cenné znalosti.

Hlavním cílem bakalářské práce je analyzovat možnosti implementace Jump serveru na platformě OpenStack. Jump server by měl poskytovat SSH tunel pro studenty, jenž se mají připojovat do instancí, které jsou virtualizovány specifickým scénářem v Kybernetické aréně. Studenti by tak mohli využívat Kybernetickou arénu pro získání cenných vědomostí, či zkušeností.

Bakalářská práce se člení na dvě kapitoly. První kapitola vysvětluje teorii problematiky virtualizace a kontejnerizace, s čím souvisí vybrání možnosti implementace Jump serveru. Taktéž je zde popsána teorie samotného OpenStacku a jeho funkcí.

Druhá kapitola se skládá z praktické části, kde na základě teoretických vědomostí je prováděna implementace Jump serveru. Ze začátku je server implementován jako kontejner, ale je zde problém v možnosti nastavení kontejneru, tudíž je Jump server implementován jako virtuální stroj.

Na základě manuálního ověření Jump serveru je server automatizován se scénáři na platformě OpenStack. Automatizace probíhá automatizačním nástrojem Ansible. Na závěr je ověřena funkčnost automatizace Jump serveru.

1 Teoretická část

V teoretické části práce je popsána virtualizace a kontejnerizace. Poté jsou vysvětleny pojmy související se zmíněnou problematikou. Následně je popsán OpenStack a jeho služby které umožňuje používat. Na závěr je provedeno porovnání VMs a kontejnerů, jejichž výsledkem bude implementace Jump serveru.

1.1 Virtualizace

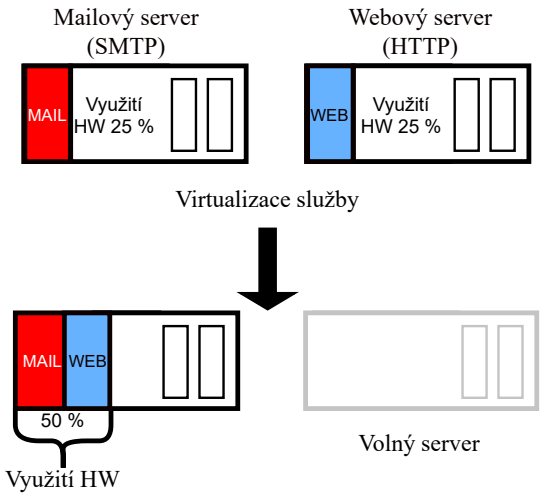
První vývoj virtualizace vznikal od 70. let 20. století firmou IBM za účelem plného využití počítačových zdrojů. Firma tak dokázala redukovat náklady za výpočetní kapacity a plně využít hardware počítače. Později v 90. letech firma VMware vyřešila možnost virtualizace pro hardwarovou počítačovou platformu x86[1]. Nyní je firma VMware globálním lídrem ve virtualizaci na platformě x86 s mnoha zákazníky [1].

Virtualizace je technologie umožňující vytvářet virtuální prostředí z fyzických počítačových zdrojů jako jsou: procesory, operační paměti, datová úložiště nebo počítačové sítě. Na fyzickém systému běží virtuální stroje (VMs), kde má každý VM svůj operační systém a chová se jako nezávislý počítač využívající fyzické zdroje hostitele. Na hostitelském hardwaru je hypervisor, který umožňuje vytvářet a spravovat již zmíněné VMs. Z praktického hlediska si lze představit 2 servery (viz obrázek 1.1). Každá služba ze serverů využívá 25 % výpočetní kapacity. S virtualizací je možné rozdělit výpočetní kapacity jednoho serveru tak, aby provozoval více služeb. Nevyužitý server může být vyřazen, popřípadě použit jako záložní. [2].

Virtualizaci lze rozdělit do několika typů na základě její využitelnosti [3, 4]:

- **Desktopová virtualizace** – slouží k oddělení fyzického počítače od prostředí stolního počítače. Virtuální prostředí lze využívat stejným způsobem, jako by se jednalo o fyzické prostředí. Pro správce je snadná aktualizace, konfigurace, údržba a zabezpečení virtuálních ploch, jelikož jsou umístěny na stejném HW. Taktéž je možné vzdálené přihlášení k virtuální ploše.
- **Virtualizace operačního systému (OS)** – je jedna z nejběžněji používaných virtualizací. Na jednom operačním systému lze současně spustit jiný OS, popřípadě více operačních systémů, dokud nejsou vypotřebovány HW kapacity. V rámci virtualizaci OS je zvýšeno zabezpečení, jelikož virtualizovaný OS je izolován, a může být snadno monitorován.
- **Virtualizace sítě** – je populární v telekomunikačním průmyslu, jelikož redukuje počet fyzických komponentů jako jsou: směrovače, přepínače, servery a kabely. Lze vytvořit několik virtuálních sítí připojených do jedné fyzické sítě.

[1] Pojem x86 označuje typ architektury procesorů, za předpokladu, že daný procesor je minimálně 32 bitový.



Obr. 1.1: Virtualizace služeb na jeden server

- **Virtualizace serveru** – k virtualizaci serveru odpovídá případ znázorněný na obrázku (1.1). Server je navržen tak, aby byly poskytovány jednotlivé služby. Služba ale nemusí využívat plně kapacitu HW serveru. Za pomocí virtualizace lze na jednom serveru poskytovat více služeb, a tak hodnotněji využít zdroj serveru.

Bez virtualizace by se mnoho firem neobešlo, jelikož přináší mnoho výhod a jednoduchých řešení problémů v oblasti telekomunikačních systémů. Hlavní výhody virtualizace jsou [5]:

- **Snížení výdajů a optimalizace fyzických zdrojů** – virtualizací lze efektivně nakládat s výpočetní technikou. Je možné využít maximální výpočetní kapacity fyzického HW. Na jednom serveru je možné virtualizací více služeb. Snížením fyzických serverů se zmenší spotřeba el. energie za provoz a údržbu.
- **Údržba** – je snadnější spravovat VMs než fyzické stroje. V případě, kdy dojde k výpadku služby ve VM, je možné odstranit výpadek snadněji než při výpadku celého fyzického stroje. Není potřeba rozsáhlý personál, tak jak v případě pro obsluhu fyzických serverů. Ve virtuálním prostředím je snadné aktualizovat a pozorovat VMs.
- **Bezpečnost** – jedna z hlavních výhod a často i jeden z důvodů zřizování VMs je bezpečnost. Všechny VMs běží ve svém sandboxovém prostředí a jsou vzájemně odděleny. Administrátor může rozhodnout k jakým zdrojům hostitele budou mít VMs přístup. V rámci bezpečnosti lze otvírat podezřelé webové stránky, popřípadě používat SW od nedůvěryhodného zdroje, aniž by došlo

k poškození systému hostitele.

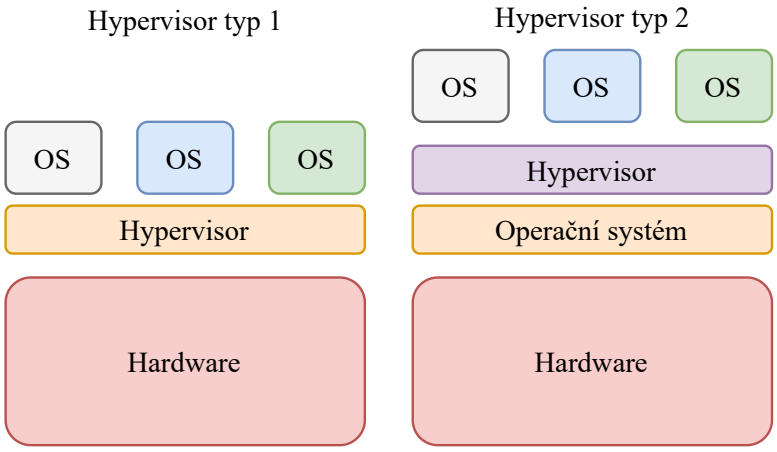
- **Záloha a testování** – VMs je možné využít v rámci testování nových aplikací, jelikož umožňují vytváření záloh. Uživatel nad těmito vytvořenými zálohami má systematický přehled a v případě poškození OS na VM, je možné obnovit původní stav VM ze zálohy. V případě poškození OS na VM hostitelský OS není nijak ohrožen.

1.1.1 Hypervisor

Hypervisor je softwarová virtualizační platforma pro správu, řízení, pozorování a vytváření VMs. Taktéž slouží k přerozdělování fyzických zdrojů hostitele pro VMs. Podle svého umístění se rozděluje na dva typy 1.2 [6].

Prvním typem je hypervisor umístěný na samotném fyzickém HW hostitele. Značí se jako tzv. „bare metal“ a nejběžněji se využívá u serverů. Slouží k řízení a monitorování hostovaných operačních systémů. Systém běží na samostatné úrovni nad hypervisorem. Příkladem pro implementaci VMs jsou: Oracle VM, Microsoft Hyper-V, VMWare ESX, Xen a KVM [7].

Druhým typem je tzv. „hosted“, který běží na OS hostitele. Hypervisor druhého typu přidává softwarovou vrstvu nad hostitelský OS. Mezi nejznámější hypervisory druhého typu patří: Oracle VM VirtualBox, VMWare Workstation a QEMU [7].



Obr. 1.2: Architektura hypervisoru

Taktéž existuje jeden nespecifický typ hypervisoru, který může být označen jako tzv. hybridní hypervisor. Tento Hybridní hypervisor se skládá z hypervisoru 1. typu KVM a 2. typu QEMU a může být označen jako KVM-QEMU. K řízení hypervisoru

KVM-QEMU je nutné pořídit nástroj Libvirt, jenž všeobecně slouží k řízeni hypervisorů, popřípadě aplikací. KVM-QEMU s Libvirtem je používán jako hypervisor OpenStacku, kde je Libvirt nainstalován nad OS [8].

1.1.2 Libvirt

Nejpoužívanější virtualizační nástroj je Libvirt, který se používá v linuxovém prostředí. Přesněji je Libvirt open-source API[2] pro správu hostovaných běžících operačních systémů na hostitelském OS. Libvirt je rozšiřující nástroj pro správu hypervisorů prvního typu. A jak zaznělo výše, je využit při správě kombinovaného hypervisoru KVM-QEMU, jenž se skládá z 1. typu KVM a 2. typu QEMU. Tento hypervisor je využíván pro cloudové[3] služby [8].

1.2 Kontejnerizace

Kontejnerizace je další možnost virtualizace za použití menšího množství výpočetních zdrojů hostitele. Ke kontejnerizaci se používá kontejner, který pro svou činnost využívá jádro operačního systému hostitele. Skládá se ze softwarového kódu a knihoven umožňující spuštění na jádře OS, na němž běží [9].

Kontejnerizace je vhodná pro vývojáře, poněvadž je zdrojový kód aplikace pohromadě s konfiguračními soubory v jednom balíčku v tzv. kontejneru. Implementace aplikace kontejnerizací je pro vývojáře snadnější než při virtualizaci. Při využití virtualizace je aplikace vytvářena ve specifickém prostředí a dochází k problémům už při implementaci na VMs pro testování. Kontejner je snadno aplikovatelný a provozovatelný na jakékoliv platformě. Při kontejnerizaci se například využívá technologie LXC, která pomocí funkcí „namespace“ nebo „cgroups“ vytváří kontejnery [10, 11]. Podrobnější popis v podkapitole 1.2.1.

1.2.1 Kontejner

Pro kontejnerizaci je základní prvek kontejner, tak jak pro virtualizaci virtuální stroj. Pro vytvoření VM je využit hypervisor a v případě kontejnerizace „Container Runtimes“ v překladu kontejnerová běhová prostředí. Běhové prostředí kontejneru jsou SW či funkce, které vytvářejí, spravují nebo spouští kontejner [12]. Tato běhová prostředí se dělí na dva typy podle jejich možností správy a tvorby kontejneru [13]:

[2]Znění zkratky API je Application Programming Interface, kde API je prostředník pro vzájemnou komunikací dvou aplikací.

[3]Cloud je označení pro servery, ke kterým je možný vzdálený přístup například webovým prohlížečem a poskytují uživatelům počítačové zdroje: úložiště, výpočetní výkon či aplikace.

- **Low-level (nízkoúrovňová) prostředí** – jsou omezena vybranými funkcemi. Především jsou zodpovědná za spuštění kontejneru. Nízkoúrovňová prostředí jsou využita u vysokoúrovňových prostředí. Typickým příkladem je využití příkazu `runc`.
- **High-level (vysokoúrovňová) prostředí** – využívají nízkoúrovňová prostředí pro tvorbu kontejnerů. Jsou zodpovědná za zpracování a přenos kontejnerových obrazů ke spuštění pro nizkoúrovňové prostředí. Typickým příkladem je běhové prostředí `containerd`, který je využíván jednou ze známějších aplikací jako je Docker.

Kontejner je forma virtualizace operačního systému. Uvnitř kontejneru jsou: binární kód, knihovny, spustitelné soubory a konfigurační soubory, které jsou právě izolovány od OS hostitele kontejnerem[14]. K izolaci se využívají dva rozdílné mechanismy, obsažené v jádře operačního systému [12, 15]:

- **Namespace (jmenný prostor)** – je mechanismus, který izoluje zdroje. Především jsou izolovány systémové zdroje, jako jsou síťová rozhraní, souborový systém, PID procesů, hostitelská a doménová jména a ID uživatelů.
- **Cgroups (kontrolní skupiny)** – se využívají pro omezení spotřeby HW zdrojů pro kontejner. Taktéž slouží pro monitorování a organizaci procesů. Hlavně přerozdělují a kontrolují spotřebu využití CPU a RAM paměti.

K vytvoření kontejneru je taktéž důležitý tzv. Container Image (diskový obraz kontejneru). V diskovém obrazu kontejneru je obsaženo běhové prostředí aplikace, které je složeno z binárních souborů a knihoven pro možné spuštění a běh kontejneru. Výhodou kontejnerových obrazů je jejich přenositelnost mezi systémy. Avšak pouze mezi systémy se stejným jádrem operačního systému, ve kterém byly vytvořeny. Z linuxového prostředí Ubuntu lze tedy spustit kontejner v dalších linuxových OS (CentOS, Fedora). V případě přenosu na OS Windows je kontejnerový obraz z linuxového prostředí nekompatibilní [13, 7].

1.2.2 Docker

Docker je jedna z nejběžněji používaných open-source aplikací pro kontejnerizaci. Aplikace umožňuje tvorbu a správu kontejnerů. Hlavním prvkem kontejneru je Docker-File (dockerový soubor), jednoduchý textový soubor obsahující příkazy k vytvoření kontejnerového docker obrazu. Docker obraz je tvořen vrstvami, které vznikají při jakýchkoliv změnách obrazu. Tyto vrstvy je možné využít pro návrat obrazu do stavu před provedenou změnou. Je možné docker obraz vytvořit úplně od začátku, ale většinou se využívá možnost stažení obrazu, který se následně upraví podle potřeb funkce kontejneru [16]. Ke stažení obrazů je k dispozici tzv. Docker Hub.

Ducker Hub je úložiště, které obsahuje přes sto tisíc obrazů pro kontejnery. Obrazy na úložišti pochází nejen od oficiálních firem či vývojářů, ale také od komunity. Podle typu předplatného je uživatel Docker Hubu oprávněn využívat úložiště pro sdílení svých vytvořených kontejnerových obrazů.

1.3 Platforma OpenStack

Openstack je open source SW vytvořený a stále spravovaný komunitou a firmami jako je například RedHat nebo IBM. Platforma je využita pro vytváření a správu veřejných a privátních cloudů.

Openstack je tvořen službami, nazývané „projekty“, které mají svoji specifickou funkci. Především slouží k obstarání základních služeb pro cloud computing, jako jsou informační výpočetní zdroje, úložiště a síťování.

Platforma Openstack je využita pro Kybernetickou arénu sloužící pro výzkum, testování a edukaci v oblasti kybernetické bezpečnosti [17]. Celá platforma je obsáhle strukturována a složena z několika funkcí, služeb či nástrojů. Služby sloužící k implementaci a testování Jump serveru jsou popsány v následujících podkapitolách.

1.3.1 Horizon

Služba horizon poskytuje uživateli grafické webové rozhraní, které je využíváno k používání služeb, jenž jsou poskytovány v OpenStacku. Ve vytvořeném projektu OpenStacku spravuje API pro samotný přístup uživatelům do prostředí. Dalo by se říci, že je to řídicí panel pro uživatele, aby mohli maximálně používat dostupné služby ve webovém rozhraní OpenStacku.

1.3.2 Neutron

Neutron je jedna z pěti hlavních služeb pro chod platformy OpenStacku. Neutron poskytuje síťové služby, které lze použít k síťování mezi virtuálními sítěmi, v kterých jsou tvořeny instance, jež jsou spravovány další hlavní službou Nova. V originálním znění lze Neutron popsat jako poskytovatele “network connectivity as a service”. Služba poskytuje uživatelům API pro nadefinování síťového připojení a adresování v cloudu. Slouží k vytváření a ke správě virtuální síťové infrastruktury. Pokročilejší funkce Neutronu je možné využít pro VPN (Virtual Private Network – Virtuální privátní síť) nebo firewally [18].

V projektu OpenStacku Neutron představuje záložku **Síť**, která slouží pro nastavení sítě. Skládá se z částí:

- **Topologie sítě** – grafické zobrazení sítí zaznamenané jako topologie nebo graf včetně zobrazení zařízení připojených v síti.
- **Sítě** – slouží pro správu sítí. Sítě lze mazat, vytvářet a upravovat již vytvořené sítě. V případě vytváření nové sítě se zadává název a následně parametry přidružené podsítě k síti. V parametrech podsítě se udává název podsítě, zdroj síťové adresy, IP adresu, verzi IP a výchozí bránu.
- **Routery** – vytváří virtuální směrovače, které splňují funkce jako fyzický směrovač.
- **Bezpečnostní skupiny** – jsou tvořeny pravidly pro IP filtraci komunikace v síti, které se aplikují na vytvořené instance a definují jim přístup k síti. Každý projekt má výchozí bezpečnostní skupinu, která se aplikuje pro jakoukoliv instanci, pokud není určeno jinak.
- **Plovoucí IP adresy** – je záložka pro přiřazování plovoucích IP adres, které jsou pořízeny podle nastavení od fyzického serveru, který spravuje administrátor cloudu. Slouží pro instance, aby se zpřístupnily pro uživatele, popřípadě aby jim bylo umožněno používat internetové připojení.
- **Firewall Groups (firewall skupiny)** – používají iptables pro tvorbu pravidel firewallu, pro každý projekt jsou vytvořena výchozí pravidla. Je možné je změnit a zároveň je aplikovat na porty směrovačů nebo VMs.
- **VPN** – poskytuje zabezpečené propojení mezi dvěma sítěmi.

1.3.3 Nova

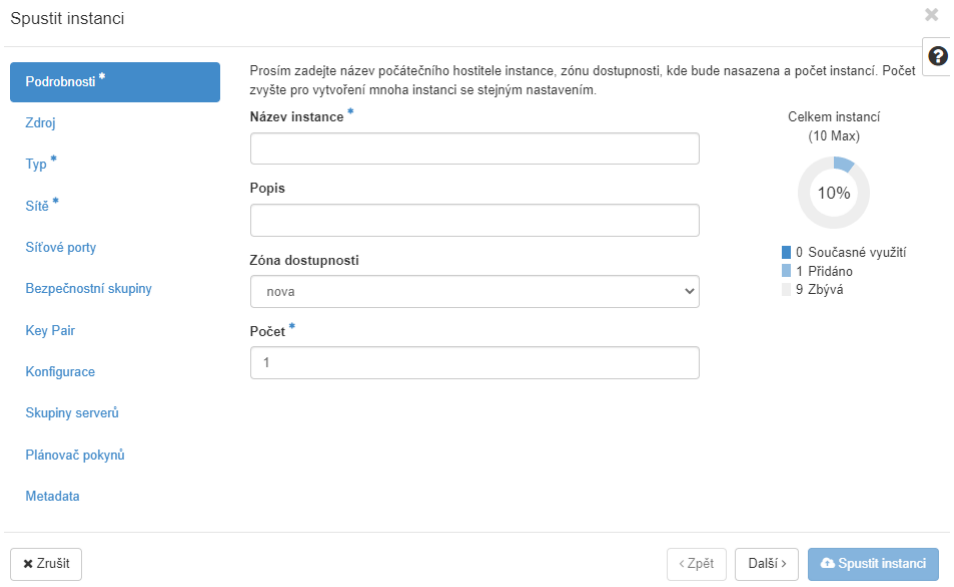
Nova projekt je služba, která podporuje vytváření a správu VMs. V případě Neutronu, kde služba slouží především pro vytváření a správu sítí, tak služba Nova slouží pro výpočetní služby OpenStack. V rámci výpočetní služby nova poskytuje daemony[4] pojmenované `nova-*`. Příkladem deamonu je `nova-compute` ve výpočetním uzlu a v řídícím uzlu je `nova-api`. Nova spravuje flavors, které definují výpočetní, paměťovou a úložnou kapacitu instancí a také mohou definovat výpočetní kapacity virtuálního serveru [19]. V povinných parametrech se udává flavor ID, název, počet vCPU, velikost paměti RAM a velikost disku. V OpenStack projektu Nova je řízena Horizonem přes sekci Compute, která se skládá z částí:

- **Přehled** – slouží pro zobrazení využití prostředků projektu. Například je zde vypsáno využití virtuálních CPU, paměti RAM, instancí, routerů, IP adres apod. Taktéž ukazuje zbývající dostupné zdroje pro vytváření virtuálních infrastruktur.

[4]Jako daemony jsou označovány softwarové programy běžící nezávisle na uživateli a často jsou automaticky spuštěny při startu systému.

- **Instance** – je vypsaný seznam vytvořených instancí (VMs), kde je možné například vyčíst název instance a obrazu, IP adresu, flavor, status atd. Také se zde vytváří samotné instance.
- **Obrazy** – seznam diskových obrazů s OS pro tvoření instancí. Je nutné je do OpenStacku přidat.
- **Klíče** – jsou tvořeny pro přístup k instancím, na výběr je z SSH klíče, nebo klíč podle X509 certifikátu. V případě SSH šifrování má klíč formát RSA.
- **Skupiny serverů** – je možnost spojování serverů do skupiny podle jejich nastavené politiky.

Při vytváření instance je nutné projít přes části: **Projekt** > **Compute** > **Instance** > **Spustit instanci** a po rozkliknutí se otevře formulář 1.3.



Obr. 1.3: Vytváření nové instance

Ve formuláři je nutné vyplnit následující části pro vytvoření instance:

- **Podrobnosti** – je důležité zadat název instance a počet instancí, které budou tvořeny se stejným nastavením.
- **Zdroj** – slouží pro výběr z jakého zdroje bude tvořena instance, je zde na výběr ze čtyř možností: obrazu, snímku obrazu, svazku nebo snímku svazku. Následně je nutné přidělit zvolený zdroj, například předem připravený diskový obraz ve formátu qcow2.
- **Typ** – přidělení instanci výpočetní, pamětní a úložné zdroje za pomocí předvytvořeného flavoru.
- **Sítě** – je nutné připojit instanci k síti. Místo sítě je možnost využít síťového portu.

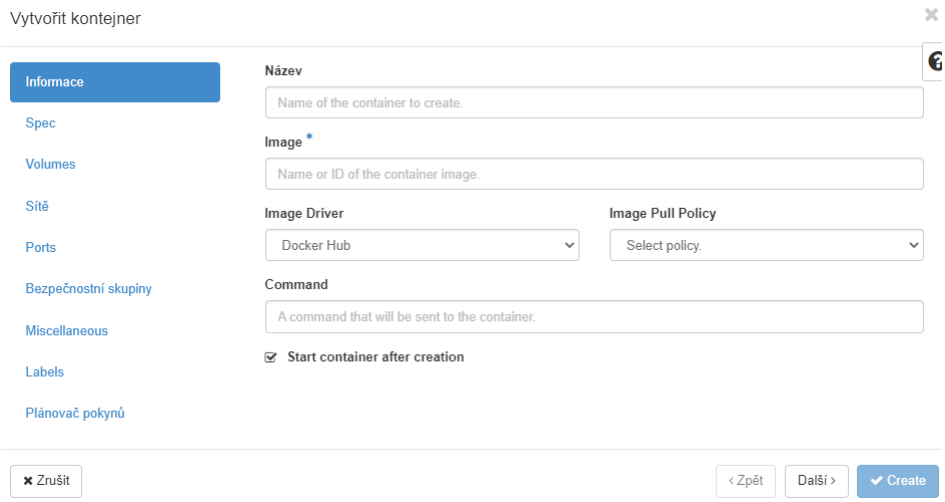
- **Bezpečnostní skupiny** – pro spuštění je nutné přidělit bezpečnostní skupinu, jež je vytvářena službou Neutron pro určení pravidel síťového provozu.
- **Key Pair** – možnost vytvořit, nebo přidělit vytvořený klíč instanci, v případě nutnosti se připojit k instanci pomocí SSH.

Další části slouží pro pokročilejší nastavení instance a nejsou důležité pro standardní tvorbu VM.

1.3.4 Zun

Zun je OpenStack kontejnerová služba pro spuštění a správu kontejnerů. S dalšími službami je možné využití rozšířit. Zun požaduje alespoň dva uzly ke spuštění kontejneru: kontrolní, který spouští identifikační a obrazové služby nástroje Zun a výpočtový, jenž spouští služby Zun pro obsluhu kontejnerů. Ve výchozím nastavení Zun pro správu kontejnerů využívá nástroj Docker [20].

Při vytváření kontejnerů je nutné projít přes části: **Projekt** > **Container** > **Kontejnery** > **Vytvořit kontejner** a po rozkliknutí se otevře formulář 1.4.



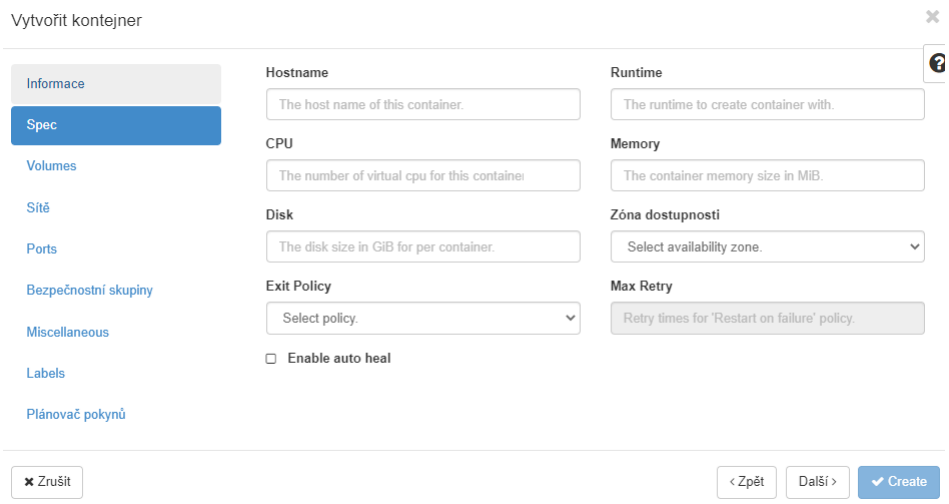
Obr. 1.4: Formulář k vytváření kontejnerů

V záložce informace je možné vyplnit:

- **Název** – je použit k přidělení jména pro kontejner, pokud není zadán, je automaticky doplněn.
- **Image** – zde je nutné zadat název nebo ID kontejnerového obrazu, který bude použit pro kontejner.
- **Image Driver** – pole pro výběr úložiště vybraného obrazu odkud bude vybraný obraz stáhnut.

- **Image Pull Police** – zde je možné vybrat volbu pro stáhnutí obrazu. Na výběr je ze tří možností:
  - **If not present** – obraz bude stáhnut, pod podmínkou, že není na lokálním úložišti OpenStacku.
  - **Always** – obraz bude vždy stáhnut pro daný kontejner.
  - **Never** – obraz nikdy nebude stáhnut a je využito pouze lokálního úložiště.
- **Command** – se využívá, pokud uživatel potřebuje spustit v kontejneru příkaz. Například po zapnutí kontejneru ihned spustit script, nebo příkaz pro stáhnutí služby.

Další záložka **Spec** 1.5 slouží pro upřesnění některých specifikací kontejneru, tyto konfigurace jsou viditelné při rozkliknutí daného kontejneru v sekci **Spec**. Slouží pro upřesnění nastavení uživatele, pokud není specifikováno, jsou hodnoty automaticky přiřazeny OpenStackem.



Obr. 1.5: Formulář k upřesnění specifikací kontejneru

- **Hostname** – slouží pro upřesnění názvu hosta. Při nevyplnění je automaticky vyplněno a je například ve formátu d633dee5f01c. Tento údaj je možné srovnat s případem názvu uživatele v kontejneru.
- **Runtime** – podle nastavení OpenStacku si mohou uživatelé zvolit typ běhového prostředí pro kontejner. Například runc nebo kata-container. Pokud pole není vyplněno, je použito výchozí běhové prostředí.
- **CPU** – přiřazuje virtuální CPU pro kontejner.
- **Memory** – přiřazuje kapacitu RAM paměti.
- **Disk** – přiřazuje velikost disku pro kontejner.
- **Zóna dostupnosti** – výběr, pro jakou zónu bude kontejner dostupný.

- **Exit Policy** – slouží k výběru výstupního pravidla pro kontejner. Jak se kontejner zachová pří výstupu z něj. Například je možné nastavit, že se odstraní, restartuje či bude restartován, pokud se stopne.
- **Max Retry** – nastavuje interval restartování kontejneru, pokud je zvoleno pravidlo restartu při selhání kontejneru.

K úspěšnému vytvoření kontejneru je nutné přidělit síť, popřípadě sítě. Přidělení sítí pro kontejner funguje obdobně, jak přidělování sítí v případě vytváření instance.

1.3.5 Heat

Heat je služba pro orchestraci cloudových aplikací za pomocí šablony vytvořené v prostředí OpenStacku. Šablony popisují infrastrukturu cloudové aplikace, která je popsána v podobě kódu jazykem YALM[5]. Přesněji šablony specifikují vztahy mezi zdroji. Taktéž je lze použít pro vytváření většiny prostředků v OpenStacku např: instancí, plovoucích IP adres, svazků, skupin zabezpečení, uživatelů atd. Heat primárně umožňuje implementovat YALM šablonami různé infrastruktury. Šablony je možné integrovat s nástroji pro konfiguraci SW, jako je Puppet nebo Ansible [21].

1.3.6 Kuryr

Kuryr je spíše plugin než služba v OpenStacku. Plugin dokáže propojovat OpenStack službu Neutron s Docker kontejner. Propojením je poskytnuta podpora síťových služeb pro Docker kontejenerem, a to poskytuje možnost připojit kontejnery do stejné sítě jako VMs. K interakci mezi kontejnery a VMs slouží Kuryr-libnetwork, to platí pro kontejnery vytvořené pod službou Zun a VMs pod Nova. Kontejnerům je taktéž poskytnuto využití služeb Neutronu, mezi ně patří například: Bezpečnostní skupiny, NAT, QoS [22].

1.4 Porovnání VMs a kontejnerů

Vzhledem k požadavkům pro jump server je nutné provést přibližnější porovnání VMs a kontejnerů. Jump server musí splňovat požadavky:

- Co nejmenší využití HW zdrojů (CPU, paměť RAM).
- Zabírat nízkou kapacitu disku.
- Podpora služeb pro konektivitu mezi fyzickou a virtuální sítí například pomocí SSH.
- Umožňovat snadnou implementaci na fyzickou síť.
- Integrace s vytvořenými šablony Heatu.

[5]YALM je typ jazyka pro serializaci dat.

1.4.1 Virtuální stroj

Při použití VM dochází k virtualizaci celého počítačového OS s tím, že je možné zvolit virtualizovaný OS. Obrazy OS pro VM jsou oproti kontejnerovým větší a trvá mnohem déle jejich stažení z kontroléru na výpočtový uzel. Většina VMs požaduje pro samotný chod minimálně 1 virtuální CPU a 1024 MB paměti RAM. V případě kapacity disku je nutné alokovat alespoň 5–10 GB, záleží na použitém diskovém obrazu. A při přenášení aplikací mezi VMs může docházet k nekompatibilitě. Aplikace běžící ve VMs jsou vzájemně izolovány. Vzhledem k bezpečnosti VMs běží izolovaně nad operačním systémem hostitele pomocí hypervisoru, ale stále je nutný přístup ke HW zdrojům hostitele: CPU, paměť RAM a disk.

1.4.2 Kontejner

Kontejnerem se virtualizuje pouze OS hostitele, kde kontejner běží na OS hostitele. Spuštění kontejneru lze docílit do pár sekund. V případě přerozdělování HW zdrojů je možné přidělit pouze podíl virtuálního CPU například 0,2 vCPU a minimálně je nutné přiřadit 6 MB paměti RAM [23]. Velikost kontejneru na disku může zabírat pár desítek MB, záleží na zvolení diskového obrazu pro kontejner. Kontejnerizované aplikace, služby se snadněji implementují mezi zařízeními a napomáhá tomu i sdílení kontejnerových diskových obrazů pomocí soukromého nebo veřejného repositáře. Je možné využít vzájemné kompatibility kontejnerů pro sdílení funkcí aplikací. Z bezpečnostního hlediska jsou kontejnery považovány za méně bezpečné než VMs, jelikož běží na OS hostitele. Toto bezpečnostní riziko však lze správnou konfigurací bezpečnostními prvky vyřešit, a v případě nutnosti je možné využít běhového sandboxového prostření (Sandboxed runtime) pro tvorbu izolovanějších kontejnerů [24].

1.4.3 Výstup porovnání

Jump server je využit k přístupu uživatelů do vzdálené privátní virtuální sítě. Pro vzdálené připojení je použita služba SSH. Vzhledem k funkci Jump serveru není potřeba vytvářet druhotný OS a je třeba zachovat co nejmenší využití kapacity HW zdrojů zařízení, na kterém bude zprovozněn Jump server. Vzhledem k požadavkům Jump serveru a výsledku porovnání (stručně v tabulce 1.1) VMs a kontejnerů, bude lepší volbou implementace kontejneru pro Jump server.

	VM	Kontejner
OS	Vlastní OS	OS hostitele
HW zdroje	Vysoké využití HW	Nízké využití HW
Využití místa na disku	5-10 GB	Několik desítek MB
Doba spuštění	Desítek sekund	Do pár sekund
Služby pro Jump server	ANO	ANO
Bezpečnost	ANO	ANO

Tab. 1.1: Stručné porovnání VM a kontejneru

2 Praktická část

V praktické části práce je popsán postup k přípravě Jump serveru, který je implementován jako kontejner. Proto je dále popsán samotný postup k tvorbě vlastního obrazu pro kontejner. Následně je z obrazu vytvořen na platformě OpenStack kontejnerový Jump server, jenž je následně testován na scénářích.

2.1 Nástroje OpenStacku

Pro vytváření experimentálního prostředí v Openstacku je využíván virtualizační nástroj Libvirt a pro kontejnerizaci je využíván Docker. V této kapitole je popsána práce se základními příkazy pro Libvirt a Docker.

2.1.1 Využití Libvirtu

K vytvoření VMs je možné využít předem vytvořený obraz VM ve formátu qcow2[1]. V případě potřeby vlastního nastavení VM lze použít diskový obraz OS ve formátu iso, který při instalaci VM vytvoří v nadefinovaném adresáři zmíněný formát disku qcow2.

Při vytvoření VM je použit diskový obraz qcow2 s linuxovým systémem Debian, který je možné stáhnout přímo od distributora[2].

```
virt-install \
--name debianVM \ #název virtuálního stroje
--memory 2048 \ #přiřazení operační paměti
--vcpus 2 \ #přiřazení procesoru
--disk path=/var/lib/libvirt/images/debian10.qcow2 #adresář obrazu k vytvoření VM
--import \ #příkaz pro import VM obrazu
--os-variant debian10 #definování instalovaného operačního systému VM
--network default #nastavení sítě VM
```

Je možné zobrazit nainstalované VMs. Podle upřesnění příkazu jsou vypsány v příkazovém řádku aktivní nebo všechny VMs. Ve výpisu je zobrazeno ID, název a stav.

```
virsh list --all #výpis všech VMs
Id Name State
 ----------------------
1 debianVM running
2 CentOS8 shut off
```

Pro potřeby podrobnějšího výpisu informací o určité VM je příkaz:

```
virsh dominfo debianVM
```

[1]Soubor qcow2 je úložný formát diskového obrazu pro virtuální stroj a k jeho formátování je používán QCOW emulátor.

[2]https://cloud.debian.org/images/cloud/

Podobným způsobem je možné vypsat informace o hostujícím počítači k zjištění výpočetní kapacity:

```
virsh nodeinfo
```

Vstup do VM je možný přes konzoli, nebo lze využít služby SSH.

```
virsh console debianVM
```

V případě nedostačujícího přidělení výpočetní kapacity VM je možné ji navýšit např. pamětí RAM nebo virtuálním CPU.

```
virsh setvcpus debianVM --maximum 2 --config
virsh setvcpus debianVM --count 2 --config
#v případě rozšíření paměti RAM:
virsh setmaxmem debianVM 2048 --config #maximalní kapacita paměti RAM
virsh setmem debianVM --config #přidělení paměti RAM zvolené VM
```

2.1.2 Využití Dockeru

Nástroj Docker nabízí uživateli velké množství operací pro práci s kontejnery. Základní příkazy slouží k práci s Docker Hub, nebo vlastním repositářem uložených kontejnerových obrazů. Pro seznámení s nástrojem autor práce zvolil obraz z Docker Hub pro tvoření nginx serveru, který je používán pro běh webové stránky. Obraz z hubu je snadné stáhnout jednoduchým příkazem:

```
docker pull nginx
```

V případě vložení kontejnerového obrazu na úložiště slouží příkaz push. Před vložením je nutné si vytvořit vlastní obraz, popřípadě nastavit jiný tag obrazu. K vytvoření vlastního obrazu je možné využít lehce pozměněný nginx obraz. Například v něm stačí vytvořit složku a příkazem commit vytvořit nový obraz.

```
docker commit <ID KONTEJNERU> <NÁZEV OBRAZU:TAG> #TAG značí číselnou verzi obrazu
docker push <NÁZEV OBRAZU:TAG>
```

Stažené kontejnerové obrazy do OS si lze vypsat:

```
docker images
```

Příkazem run se vytvoří a spustí kontejner ze staženého obrazu.

```
docker run -p 8080:80 nginx #nastavení portu pro aplikaci a~kontejner
```

Pro kontrolu, zda server běží, je možné využít adresu http://localhost:8080 přes internetový prohlížeč. Na serveru běží defaultní webová stránka konfigurována v repositáři /usr/share/nginx/html a pojmenována jako index.html. Kontejnery si lze vypsat:

```
docker ps #aktivní kontejnery
CONTAINER ID IMAGE COMMAND CREATED STATUS
68d1bce2dbd5 nginx "/docker-entrypoint." 7 minutes ago Up 7 minutes
```

Další část výpisu:

```
docker ps #aktivní kontejnery
CREATED STATUS PORTS NAMES
7 minutes ago Up 7 minutes 0.0.0.0:8080->80/tcp, :::8080->80/tcp boring_kepler
```

V případě nutných změn přímo v kontejneru lze do něj vstoupit pomocí příkazu `exec` a ID nebo jméno kontejneru. Také je nutné zadat parametry -i a -t, jež umožňují interakci a spuštění příkazu v kontejneru. V případě příkladu je použit příkaz bash, který nás dostane dovnitř kontejneru.

```
docker exec -it 68d1bce2dbd5 /bin/bash
```

V kontejneru je uživatel přihlášen jako správce s ID kontejneru např.:

root@68d1bce2dbd5:/. Při vytváření vlastního obrazu z DockerFile je používán příkaz `build`.

```
docker build -t <NÁZEV-OBRAZU>:<TAG> "</cesta k souboru DockerFile>"
```

Pokud se nacházíme v repozitáři, kde je umístěný DockerFile, tak v případě použití `build` není třeba zadávat cestu k souburu, ale stačí použít tečku za příkazem.

```
docker build -t <NÁZEV-OBRAZU>:<TAG> .
```

Podoba textového dokumentu DockerFile:

```
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
COPY script.sh /script.sh
CMD ["/script.sh"]
```

V případě příkladu DockerFile je zvolen základní obraz nginx. Při vytváření obrazu jsou pak zkopírovány z počítače vytvořené soubory index.html a script.sh. Každá operace při vytváření kontejnerového obrazu má svůj unikátní příkaz. Popis příkazů pro základní operace v souboru DockerFile:

- **FROM** – Každý DockerFile musí začít příkazem FROM, který definuje základní obraz pro nově vytvářený kontejnerový obraz.
- **COPY** – příkaz sloužící pro kopírování souborů. Při vytváření kontejnerového obrazu je možné kopírovat do obrazu soubory z uživatelova disku.
- **RUN** – specifikuje příkazy, jež jsou spuštěny v případě sestavovaní obrazu.
- **CMD** – specifikuje příkaz nebo sadu příkazů, které budou spuštěny, při spuštění obrazu v kontejneru.
- **ADD** – příkaz podobný příkazu COPY, nejen že umožňuje kopírování souborů, adresáře, ale také i URL adres ze vzdálených souborů do souborů nově vytvářeného obrazu.
- **LABEL** – umožňuje specifikaci metadat v kontejnerovém obrazu. Data jsou psána v podobě klíč-hodnota, a jsou ukládány jako řetězce znaků.

- **MAINTAINER** – příkaz pro přidání metadat o informaci autora kontejnerového obrazu. Doporučuje se používat již zmíněný příkaz LABEL, který má více možností nastavování metadat.

2.2 Příprava Jump serveru

V této kapitole je provedena analýza obrazů, podle níž bude vybrán obraz pro samotný Jump server. Po vyhodnocení analýzy je popsáno vytvoření Jump serveru pomocí vybraného obrazu, který je použit jako základ v DockerFile. Součástí vytváření jsou popsány potřebné služby a soubory pro server.

2.2.1 Analýza obrazů

Pro výběr obrazu byla provedena analýza dostupných obrazů pro Jump server. V rámci analýzy byly porovnány obrazy z veřejně dostupného úložiště Docker Hub. Porovnané obrazy jsou zachyceny v tabulce, v níž jsou seřazeny podle jejich velikostí.

Název	Velikost	Základ obrazu
binlab/docker-bastion	6,72 MB	Alpine
connesc/ssh-gateway	8,4 MB	Alpine
pingbo/ssh-tunnel	13 MB	Alpine
n4de/sshd	14,9 MB	Alpine
kamranazeem/ssh-server	15,5 MB	Alpine
Panubo/sshd	33,7 MB	Alpine
anthonyneto/sshserver	47,3 MB	Alpine
mulinbc/sshd	431 MB	Fedora

Tab. 2.1: Příkladné porovnání kontejnerových obrazů pro Jump server

Podle porovnání v tabulce je možné vyčíst, že obrazy s nejnižší velikostí jsou tvořeny se základním obrazem Alpine Linux. Používaný obraz Alpine patří pod linuxovou distribucí a byl vytvořen pro efektivní využití HW zdrojů. Jeho velikost je okolo 5 MB. Vzhledem k jeho vlastnostem má vhodné využití jako základní obraz pro Jump server. Také je zřejmé, že je často využíván pro SSH konektivitu.

2.2.2 Tvorba vlastního kontejneru pro Jump server

Tvorba vlastního obrazu pro kontejner Jump serveru byla prováděna ve virtuálním stroji vytvořeným v OpenStacku. Pro VM byl zvolen OS debian, ve kterém byl

zprovozněn Docker. Instalace Dockeru probíhala podle návodů, které jsou veřejné na oficiálních stránkách Dockeru.

K vytvoření vlastního obrazu bylo potřeba vytvořit textový dokument DockerFile. V tomto textovém souboru byly nadefinovány potřebné konfigurace pro Jump server. Nejprve byl nadefinován základní obraz, na kterém jsou prováděny konfigurace a instalace služeb. Jak bylo dříve zmíněno, byl vybrán jako základní obraz Linux Alpine. V dalších krocích v souboru jsou nadefinovány příkazy pro stáhnutí služby SSH a nakonfigurování souboru $sshd\_config$, který definuje nastavení SSH služby. Například je nutné povolit TCP přesměrování či nastavení portu 22, který je standartně pro SSH. V případě přesměrovávání budou na serveru manuálně zvoleny jiné porty. Textový soubor DockerFile byl po nakonfigurování uložen a použit pro vytvoření obrazu. Podrobnější konfiguraci DockerFile je možné vidět níže:

```
FROM alpine:latest

RUN apk add --update --no-cache openssh bash
RUN sed -ie 's/#Port 22/Port 22/g' /etc/ssh/sshd_config
RUN sed -ri 's/#HostKey \/etc\/ssh\/ssh_host_key/HostKey \/etc\/ssh\/ssh_host_key/g'
  ↪/etc/ssh/sshd_config && \
   sed -ir 's/#HostKey \/etc\/ssh\/ssh_host_rsa_key/HostKey \/etc\/ssh\/ssh_host_rsa_key/g'
     ↪/etc/ssh/sshd_config && \
   sed -ir 's/#AllowTcpForwarding yes/AllowTcpForwarding yes/' /etc/ssh/sshd_config && \
   ssh-keygen -A && \
   ssh-keygen -t rsa -b 4096 -f /etc/ssh/ssh_host_key

RUN sed -ie 's/#PubkeyAuthentication yes/PubkeyAuthentication yes/g' /etc/ssh/sshd_config && \
   sed -ie 's/#PasswordAuthentication yes/PasswordAuthentication yes/g' /etc/ssh/sshd_config && \
   >> /etc/ssh/sshd_config

EXPOSE 22

CMD ["/bin/bash"]
```

Následujicím příkazem byl DockerFile použit pro vytvoření vlastně nakonfigurovaného obrazu. Vytvořený obraz byl pojmenován jako jumpserver.

```
docker build -t jumpserver .
```

Po vytvoření obrazu byl založen účet na Docker Hubu, aby bylo možné použít obraz pro kontejner v OpenStacku. Vytvořený účet byl použit pro přihlášení v samotném Dockeru ve VM, kde byl vytvořen obraz. Před posláním obrazu na úložiště bylo potřeba vytvořit tag existujícího obrazu pro vytvořený repozitář na úložišti a následně byl obraz vložen na zvolený repozitář úložiště.

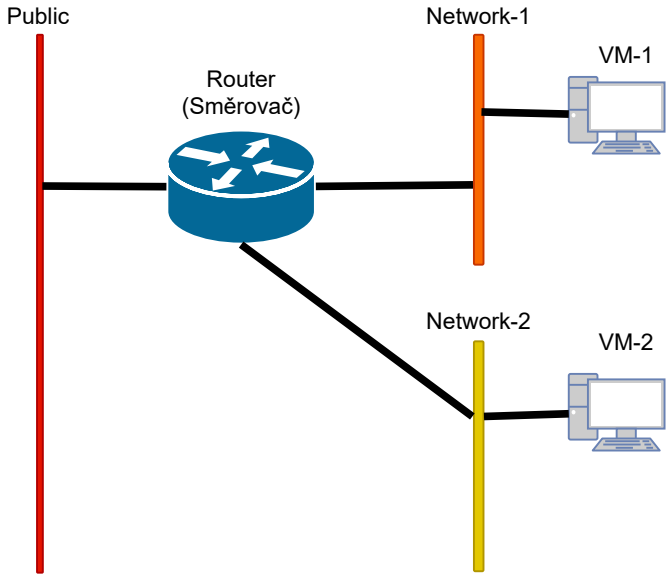
```
docker tag jumpserver xkomar31/jumpserver
docker push xkomar31/jumpserver
```

Nyní je možné obraz kontejneru stahovat přes úložiště Docker Hub, ke kterému byl automaticky přidělen příkaz pro pull.

```
docker pull xkomar31/jumpserver
```

2.3 Topologie sítě

Topologie sítě experimentálního prostředí, ve kterém bude testován Jump server se skládá ze dvou virtuálních strojů, dvou sítí a jednoho směrovače 2.1. Jump server bude poskytovat připojení k virtuálním počítačům za NAT sítí směrovače.

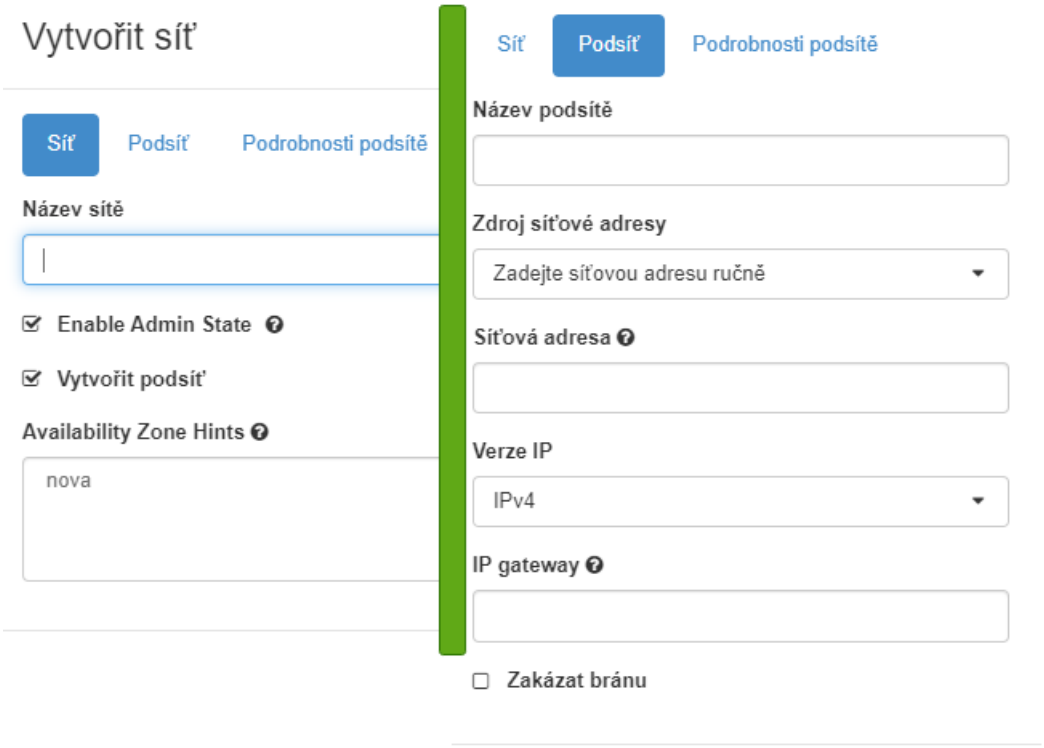


Obr. 2.1: Topologie experimentálního prostředí

2.3.1 Tvorba sítě

Byly vytvořeny sítě Network-1 a Network-2. K vytvoření jedné ze sítí v OpenStacku je nutné v záložce projekt rozkliknout síť, v ní sítě a na pravém horním rohu zvolit možnost vytvořit síť. Po provedení je otevřen formulář 2.2. Do formuláře se zadává název sítě, a zda bude vytvořena podsíť. Vzhledem k příkladu bylo do názvu sítě vepsáno Network-1 a ostatní možnosti byly ponechány ve výchozím stavu, jelikož je vytvářena podsíť pro VM-1. Při vytváření podsítě (vpravo 2.2) se zadává název podsítě, zdroj síťové adresy, síťová adresa, verze IP a IP výchozí brány. Pokud není zadána IP výchozí brány, je přidělena první IP adresa sítě. V poslední záložce

podrobnosti podsítě lze nastavit, zda bude využito služby DHCP, a jaký bude rozsah přidělovaných adres, popřípadě které adresy budou přidělovány ručně. Taktéž je možné zvolit adresy pro službu DNS. Po vyplnění stačí dát v pravém dolním rohu formuláře vytvořit a síť bude vytvořena.



Obr. 2.2: Formulář k vytváření sítí

2.3.2 Tvorba routeru

K propojení dvou vytvořených sítí je možné vytvořit router. Router se vytváří ve stejné záložce jako sítě. Pro vytvoření routeru stačí zadat název, avšak podle topologie je routeru přidělena externí síť public. Po vytvoření je nutné přidat v rozhraní routeru vytvořené podsítě sítí, ke kterým budou přiřazeny VMs.

2.3.3 Tvorba Instance

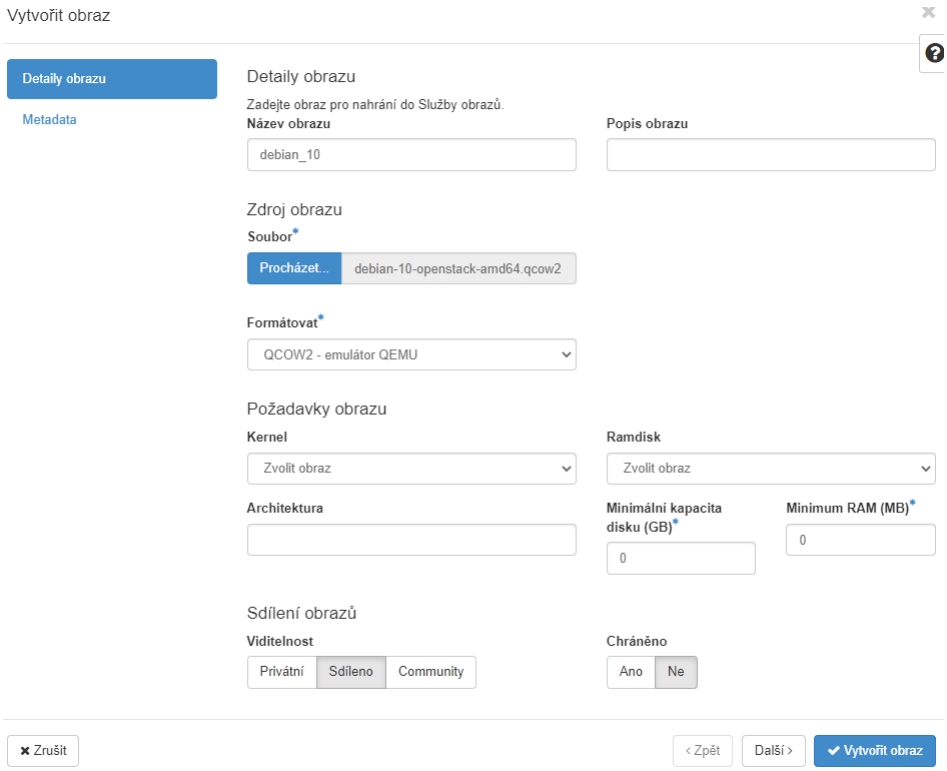
Tvorba instance je popsána v teoretické části a vyobrazena na obrázku 1.3. V praktické části jsou tvořeny dvě instance, VM-1 a VM-2. Pro VM-1 je použit diskový obraz debian-10, který byl nejdříve importován do OpenStack úložiště. U VM-2 byl vybrán obraz centos8, jenž byl sdílený na úložišti. Jako typ byl vybrán pro oba VMs

flavor s přiřazením 1 virtuálního procesoru, 1 GB RAM paměti a 10 GB místa na disku.

K vytvořeným instancím jsou připojena rozhraní dle topologie, VM-1 je připojena do podsítě sítě Network-1 a VM-2 do podsítě sítě Network-2. Následně jim jsou přiřazeny pomocí DHCP IP adresy.

Obraz pro instanci

Pro vkládání diskových obrazů do OpenStack úložiště slouží sekce Obrazy, která je umístěna pod záložkou Compute. Před implementací je doporučené stáhnout disk ve formátu qcow2, jenž je přizpůsoben pro OpenStack, který je importován do úložiště. Formulář pro přidávání obrazů je zobrazen na obrázku 2.3.



Obr. 2.3: Formulář k přidání obrazu do OpenStacku

Pro úspěšné vložení je nutné vyplnit název obrazu, zdroj a podle typu vybrat formátování. V ukázkovém případě je zvolen emulátor QEMU na qcow2 formát disku. Další nutnou položkou je vyplnění minimální kapacity disku a paměti RAM, avšak ve výsledku přidělených HW zdrojů rozhoduje typ instance při použití obrazu. Uživatel může nahraný obraz zpřístupnit vybráním viditelnosti pro ostatní uživatele projektů na platformě OpenStack. Po vyplnění je nutné vytvoření potvrdit tlačítkem vytvořit obraz.

2.3.4 Tvorba kontejneru

Při tvorbě kontejneru byly nastavovány konfigurace, jež jsou vepsány v tabulce 2.2:

Název	Jumpserver
Image	xkomar31/jumpserver
Image Driver	Docker Hub
Image Pull Policy	Ponecháno výchozí nastavení.
CPU	0,3
Memory	256 MB
Sítě	Přiděleny dle daného scénáře.

Tab. 2.2: Konfigurace kontejneru při vytváření

Sítě byly nastavovány podle potřeb k danému scénáři. Další možná nastavení zůstala ve výchozím stavu, jelikož v rámci testování funkčnosti nemají žádný vliv.

2.3.5 Příprava scénáře

Před testováním funkčnosti bylo nutné nastavit VMs pro možnost autentizace pomocí hesla. Jelikož k vytvořeným VMs je pouze povolený přístup přes SSH s ověřením totožnosti přes veřejný klíč. Díky přiřazenému klíči při vytváření VMs a přidělení plovoucích adres, je možné se k VMs připojit přes SSH.

K SSH připojení k VM bylo využito linuxového subsystému (WSL) Ubuntu pro operační systém Windows. Pomocí WSL je možné provést instalaci linuxového jádra na OS windows, jenž umožňuje používat linuxové příkazy a nástroje. Poté je možné použít linuxový příkaz pro SSH, pokud je zapnuta služba sshd. V opačném případě k připojení k VMs je možné použít PuTTY, což je klient pro protokoly SSH, Telnet atd. Po připojení na VMs bylo nastaveno heslo a v konfiguračním souboru pro SSH byla nastavena autentizace pomocí hesla. Změnu hesla je nutné provést s příkazem sudo a v souboru je nutné odkomentovat řádek a nastavit PasswordAuthentication yes.

```
sudo passwd <UŽIVATEL>
```

Na základě zvolené sítě, byla instanci přidělena IP adresa z DHCP služby. Pokud je k instanci připojeno síťové rozhraní až po vytvoření, je možné zvolit vlastní adresu v rozsahu adres sítě. V posledním případě je možné nastavit IP adresu manuálně vně VM přes konfigurační soubor, který je umístěný v /etc/network/ a je pojmenován jako interfaces. V případném scénáři byly zachovány IP adresy přiřazené DHCP službou.

2.4 Tunelování SSH

Pro instance, které jsou umístěny za 1. směrovačem je možné přiřadit plovoucí IP adresu. Plovoucí adresa pochází od fyzického serveru, díky ní mají instance přístup na internet a zároveň je možné adresu využít pro vzdálené připojení k instaci v rámci virtuální privátní sítě. Problém nastává, když instance je v topologii virtuální sítě až za druhým směrovačem. V tomto případě fyzický server nemá dosah k instanci a není možné přiřadit plovoucí adresu. Proto k takovým instancím bude využit Jump server. K tunelování přes Jump server existuje několik způsobů, přes které je možné vytvářet SSH tunel skrz Jump server k instancím. V případě praktické části je využíváno lokálního přesměrování portů skrz Jump server na VMs. V dalším případě je možné využít Jump server pro dynamické, popřípadě statické přesměrování na VMs.

Pro dynamické přesměrování je typickým příkladem využití příkazu `ssh -J`. Uživatel je nucen zadat adresu Jump serveru a adresu VM. V případě možnost `ssh -J` je vytvářeno SSH spojení prvním hostem, který může být označen jako jump host, přes kterého je přesměrován na vybranou instanci.

```
ssh -J <uzivatel@IP-JUMPSERVER> <uzivatel@IP-VM-1>
```

Tento způsob je velice nepraktický a není plně využit potenciál Jump serveru. Pro snadnější využití je možné využít statického přesměrování, kde je uživatel nucen zadat cesty skoků v /.ssh/config, taktéž je tento způsob znám jako ProxyJump. Nastavování pro jednotlivé uživatele a scénáře by bylo obtížné a nepraktické. Pro praktičtější a šetrnější řešení je využito lokálního přesměrování portů, čehož lze dosáhnout základním příkazem `ssh -L`.

Pomocí lokálního přesměrování je otevřen port Jump serveru, který přesměrovává komunikaci na vzdálenou instanci. Po připojení na otevřený port je uživatel přesměrován na cílovou instanci, pro kterou byl port otevřen. Tento způsob tunelování je vhodný, jelikož stačí otevřít porty k instancím a uživatel zadá pouze adresu serveru a port, ke které instanci má být připojen. Nejtypičtější příklad pro lokální přesměrování:

```
ssh -L <IP-JUMPSERVER>:<OTEVŘENÝ PORT>:<IP-INSTANCE>:<PORT K INSTANCI> uzivatel@<IP-INSTANCE>
```

Po zadání se server připojí na instanci, aby tomu bylo zabráněno a mohlo být otevřeno více portů pro další instance, je nutné tento provoz tunelu uvést do pozadí v Jump serveru. K tomu slouží možnost rozšíření příkazu, kde -fN je nastavení pro běh SSH připojení na pozadí, a aby nebyl prováděn žádný vzdálený příkaz, což umožňuje mít otevřené připojení v pozadí serveru.

```
ssh -fN -L
```

Jedna z dalších možností je použití pravidel v iptables k přesměrování přes port do zvolené NAT destinace. Je nutné nastavovat jednotlivé příkazy na serveru. Tento způsob by byl velice nepraktický a náročný pro nastavování pro všechny možné scénáře. Pokud by se v pravidle vyskytla chyba, tak by ji bylo obtížné najít.

2.5 Manuální otestování funkčnosti

V následujících podkapitolách je popsán postup testování funkčnosti Jump serveru. Nejprve probíhá testování SSH konektivity v rámci jedné sítě, pro ověření funkcionality SSH tunelu. V dalších krocích probíhá testování SSH pro zvolený scénář. Na závěr kapitoly je popsán výsledek z otestování funkcionality.

2.5.1 Testování funkčnosti

Při vytváření sítí byly vytvořeny i jejich přidružené podsítě. První podsíť měla možný rozsah adres 192.168.80.0/24 a druhá podsíť 192.168.90.0/24. Podle tabulky 2.3 byly instancím přiděleny následující IP adresy:

VM-1 (debian)	Network-1-subnet: 192.168.80.208
VM-2 (centos)	Network-2-subnet: 192.168.90.198

Tab. 2.3: IP adresy jednotlivých VMs

V rámci jedné sítě

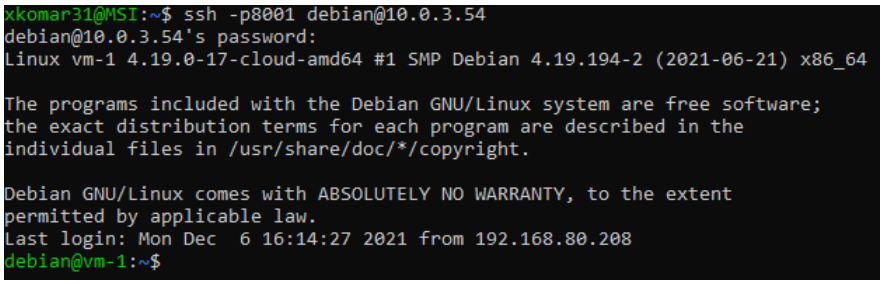
Ověření funkčnosti nejdřív proběhlo v rámci jedné sítě. Vytvořený kontejner byl přiřazen do public a Network-1 sítě. Kontejneru byly přiřazeny IP adresy z podsítě Network-1-subnet a z public sítě. Hlavní IP adresa je však přiřazena public sítí, jelikož ta je využita pro SSH přesměrování klienta k VM. Kontejner obdržel IP adresu: 10.0.3.54 Pro ověření funkčnosti bylo vytvořeno SSH spojení z kontejneru do VM-1. K otevření spojení byl použit příkaz:

```
ssh -fN -L 10.0.3.54:8001:192.168.80.208:22 debian@192.168.80.208
```

Po zadání hesla uživatele VM-1, bylo v pozadí kontejneru vytvořeno SSH spojení k VM. Klient po zadání příkazu k SSH připojení ke kontejneru na adrese 10.0.3.54 na otevřeném portu 8001 je přesměrován do VM-1.

```
ssh -p8001 debian@10.0.3.54
```

Po zadaní je klient přepojen na VM-1. Pro úspěšné připojení je nutné zadat zvolené heslo uživatele. V ukázkovém příkladě se jedná o uživatele debian. Ověření je vyobrazeno na obrázku 2.4



Obr. 2.4: Úspěšné připojení k VM-1

V rámci dvou sítí

Ověření funkčnosti v rámci zvolené topologie, která je tvořena dvěma sítěmi, byl vytvořený kontejner 2.1 přiřazen do sítě public, Network-1 a Network-2. V tomto případě kontejner obdržel od DHCP sítě public IP adresu 10.0.4.84. Poté byly nastaveny jednotlivé příkazy pro obě VMs.

```
ssh -fN -L 10.0.4.84:9001:192.168.80.208:22 debian@192.168.80.208
ssh -fN -L 10.0.4.84:9002:192.168.90.198:22 centos@192.168.80.208
```

Pro připojení k VM-1 byl na straně klienta zadán příkaz:

```
ssh -p9001 debian@10.0.4.84
```

Příkaz pro připojení k druhé VM-2:

```
ssh -p9002 centos@10.0.4.84
```

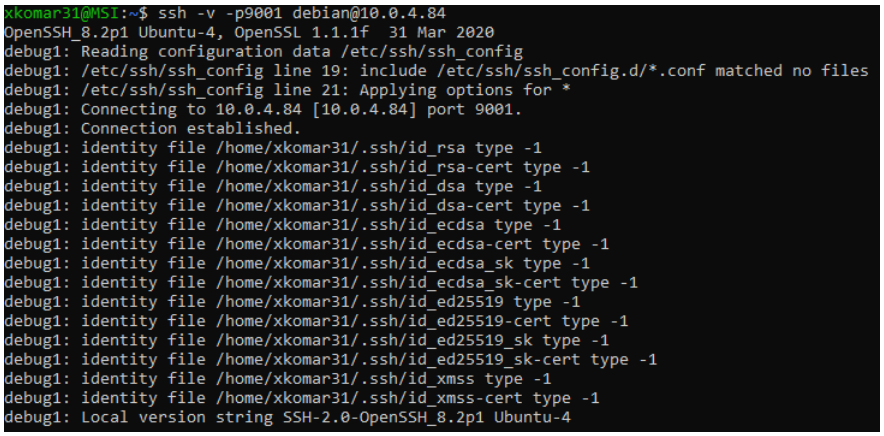
V obou případech se nepodařilo připojit k daným VM. Aby byla zjištěna příčina, byl v obou případech příkaz k připojení rozšířen o možnost -v, jež umožňuje vypsat průběh 2.5 SSH připojení. Připojení k portu vždy proběhlo správně, ale klient neobdržel zprávu v prostém textu ohledně verze protokolu od VMs.

```
ssh -v -p9001 debian@10.0.4.84
```

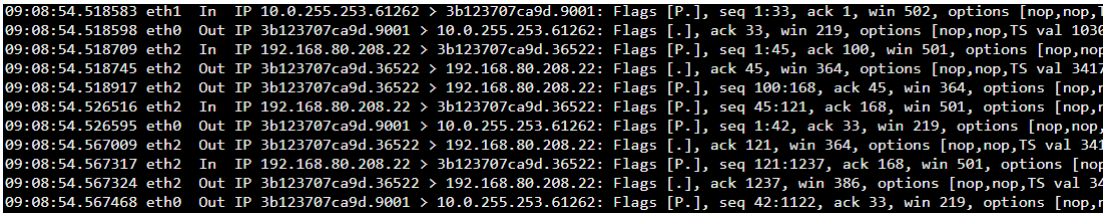
Tento incident 2.5 nastával v obou případech. Další kroky vedly k hledání chyb. Nejprve proběhl test, zda probíhá SSH propojení mezi VM a serverem. K tomuto testu byl využit příkaz tcpdump, jenž slouží pro analýzu komunikace v síti. Nejprve bylo nutné manuálně stáhnout službu do Jump serveru a poté byla provedena analýza.

```
apk add tcpdump
tcpdump -i any
```

Předchozím příkazem byl zapnut odposlech, který zachycuje komunikaci na serveru. Při odposlechu sítě bylo zjištěno, že probíhá připojení k serveru ze strany klienta, ale nepřichází přes server odpověď od VM. Tento odposlech je zachycen na obrázku 2.6. V případě úspěšného připojení, jež proběhlo v rámci jedné sítě musí vypadat

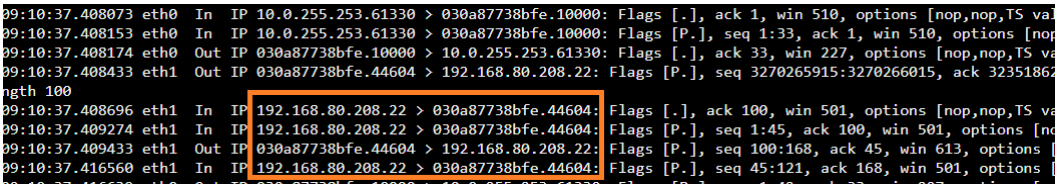


Obr. 2.5: Neúspěšné připojení



Obr. 2.6: Odposlech neúspěšného připojení

spojení jako na obrázku 2.7, kde je možné vidět komunikaci od VM, která přes Jump server předala verzi SSH protokolu.



Obr. 2.7: Odposlech úspěšného připojení

2.5.2 Výstup testování funkčnosti

Připojení k VMs přes Jump server bylo neúspěšné. Pro nalezení možných chyb byly dovytvořeny obdobné scénáře, u kterých taktéž probíhalo připojení k VMs za NAT síť přes kontejnerový Jump server. U některých scénářů bylo připojení úspěšné. Pro nalezení anomálie byly scénáře porovnávány. Hlavní problém byl vyřešen při

porovnávání směrovacích tabulek Jump serveru. V neúspěšných scénářích byla nastavena výchozí hodnota brány na některý ze směrovačů v experimentální síti. Aby bylo úspěšné připojení přes Jump server je nutné zajistit výchozí hodnotu brány nastavenou na public síť nikoliv na některých ze směrovačů dané sítě. Přenastavit výchozí bránu v kontejneru Jump serveru je samotným OpenStackem zakázané a jelikož je samotný OpenStack v neustálém vývoji povolit či přepracovat některé funkce v OpenStacku by mohlo mít špatný vliv na jeho samotný chod, popřípadě na některé jeho funkce. V případě implementace Jump serveru jako VM, je možné měnit směrování v případě stejného problému. Proto bude pro Jump server použit **virtuální stroj**.

2.6 Implementace Jump serveru jako VM

Vzhledem k výsledku ověření funkčnosti Jump serveru jako kontejner, kde nastával hlavní problém ve směrování v kontejneru, které nelze pozměnit, tak bude využita druhá varianta implementace Jump serveru, a to s využitím VM, jenž umožňuje podrobnější nastavení Jump serveru.

2.6.1 Varianty implementace

Za využití Jump serveru jako VM se naskytuje několik variant, jakým způsobem bude provedena implementace Jump serveru. Nutné zvolit tu variantu, jenž se nejvíce přibližuje již zmíněným požadavkům Jump serveru.

V první variantě, kterou je možné jen zmínit, jelikož neefektivně nakládá s výpočetní technikou OpenStacku je za využití Jump serveru pro každý scénář zvlášť, kde bude vytvořen pro každý scénář vlastní Jump server, který bude připojen do potřebných sítí scénáře.

V druhé variantě by se VM Jump serveru připojovala k sítím, v kterých jsou umístěny VMs, ke kterým je potřeba zajistit přístup SSH připojení přes server pod podmínkou, že není možné k nim přiřadit plovoucí IP adresu.

V posledním případě bude vytvořen Jump server, který bude připojen do specifické sítě, která bude využita pro přidělování IP adres a možné SSH konektivitě mezi VMs a Jump serverem. VMs bez možnosti přiřazení plovoucích IP adres budou připojeny k specifické Jump síti. Tato poslední možnost je nejvíce nejefektivnější nejen pro výpočetní kapacitu, ale také i pro automatizaci.

2.6.2 Vytvoření Jump serveru

První krok k implementaci Jump serveru jako VM vedl k vytvoření instanci, která byla připojena do sítě public, od které obdrží IP adresu, jenž je využívána pro připojení k Jump serveru z fyzické sítě, a také je přes ní prováděn SSH připojení k potřebným VMs. Jako operační systém byl zvolen z OpenStack úložiště Debian. Podrobnější konfiguraci instance je možné vidět v tabulce 2.4.

Název instance	Jump-server
Obraz instance	debian-10
Flavor	arena.1-1.5
Síť	Jump-Network

Tab. 2.4: Informace o Jump serveru

2.6.3 Vytvoření sítě pro Jump server

Jelikož byla vybrána varianta, kdy jsou potřebné VMs připojovány k Jump síti je nutné ji vytvořit. Vytvořená Jump síť je připojena přes rozhraní Openstacku k Jump serveru. Úplná konfigurace sítě je popsána v tabulce .

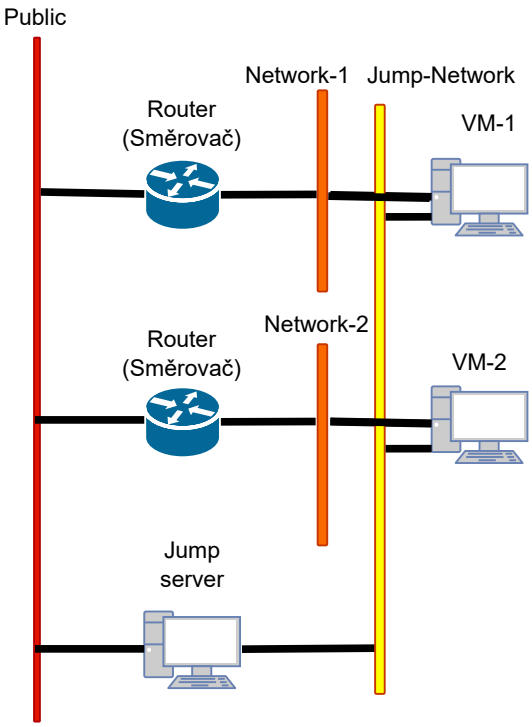
Název sítě	Jump-Network
Název podsítě	Jump-subnet
rozsah IP adres	192.168.100.0/24

Tab. 2.5: Informace o Jump síti

2.6.4 Příprava scénáře

Topologie scénáře zůstává obdobná jak při ověřování SSH konektivity Jump serveru jako kontejner. Hlavni rozdíl spočívá v tom, že kontejner Jump serveru byl připojován k sítím VMs, ale v případě VM Jump serveru jsou jednotlivé VMs topologie do již vytvořené Jump sítě. Celá topologie i s Jump serverem je vyobrazena na obrázku 2.8.

Připojené VMs do sítě obdrželi od DHCP služby IP adresy. Je možné při připojování VMs do sítě zvolit adresu z povoleného rozsahu sítě, popřípadě ji změnit v konfiguračním souboru v samotném systému VMs. Přidělené IP adresy jsou vepsány v tabulce 2.6. Je důležité mít nastavené k uživatelům VMs hesla, jelikož jsou součástí konfigurace SSH připojení. Konfigurace hesel proběhla v rámci přípravy scénáře pro Jump server kontejneru.



Obr. 2.8: Topologie s umístěným Jump serverem.

VM-1 (debian)	JumpNetwork: 192.168.100.188
VM-2 (centos)	JumpNetwork: 192.168.100.51

Tab. 2.6: IP adresy jednotlivých VMs v Jump síti

2.6.5 Manuální ověření funkčnosti

Obrazy systémů v OpenStacku pro instance mají v defaultním nastavení povolené SSH připojení pomoci SSH klíče. V případě potřeby připojení do Jump server z jiného zařízení, než byl použit SSH klíč pro instanci Jump serveru je možné taktéž nastavit heslo pro uživatele na straně serveru.

Pro vytvoření SSH tunelu mezi Jump serverem a VMs slouží obdobný příkaz jako v případě kontejneru. Za pomocí přidělených IP adres 2.6 je možné nakonfigurovat SSH připojení k VMs. Příkaz pro vytvoření SSH spojení na Jump serveru:

```
ssh -fN -L 10.0.1.253:5001:192.168.100.147:22 debian@192.168.100.147
ssh -fN -L 10.0.1.253:5002:192.168.100.188:22 centos@192.168.100.188
```

Pro zadání každého příkazu je nutné zadat heslo k jednotlivých VMs. Na zvolených portech vznikné SSH tunel k VMs přes Jump server. K připojení k VM-1 je

nutné na straně klienta zadat:

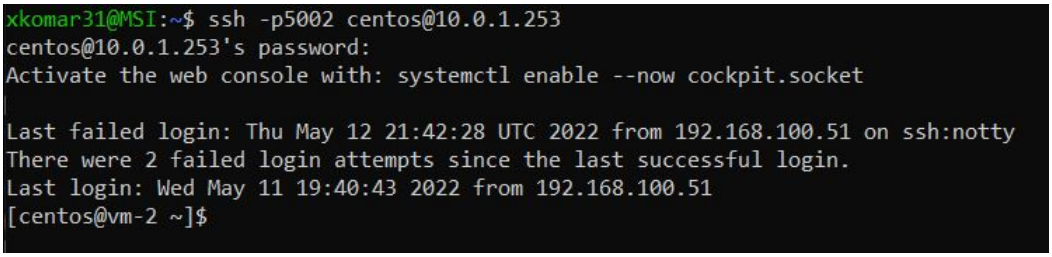
```
ssh -p5001 debian@10.0.1.253
```

V případě připojení k VM-2:

```
ssh -p5002 centos@10.0.1.253
```

Příkaz se mění pouze podle přiděleného portu k VMs a uživatele na VM. Po zadání je klient okamžitě přesměrován do systému VMs. Úspěšné připojení k VM-1 přes Jump server je možné vidět na obrázku 2.9 a v případě VM-2 2.10.

Obr. 2.9: Připojení přes Jump server k VM-1.



Obr. 2.10: Připojení přes Jump server k VM-2.

Možné problémy s SSH připojením

Je pravděpodobné, že po restartu nebo vypnutí a zapnutí Jump serveru je špatně přidělena výchozí brána a nebude možné navázat SSH připojení, ať už s Jump serverem nebo k jednotlivým VMs. Je to stejný problém jak v případě Jump serveru jako kontejner, ale v případě Jump serveru jako VM je možné změnit výchozí bránu k public síti. Příkaz pro změnu výchozí brány:

```
sudo ip route change default via 10.0.0.1
```

Výchozí brána nesmí být nastavena na kterýkoliv směrovač v síti scénáře. Zpravidla by měla být přidělena výchozí brána podle první přidané sítě do instance.

2.7 Řešení duplicitních IP adres

Scénáře Kybernetické arény budou dostupné pro více uživatelů najednou. V případě scénáře za použití Jump serveru jako kontejner by mohlo docházet k duplicitě adres v síti, jenž by vedlo k problému připojení k scénáři. Tento incident by musel být na straně serveru ošetřen, například skriptem pro úpravu IP adres.

V případě scénáře za použití Jump serveru jako instance nebude docházet k duplicitě IP adres. Instance, ke kterým má být zahájen SSH tunel ze strany serveru, jsou připojovány do Jump sítě, kde obdrží z DHCP unikátní adresu v nastaveném rozsahu. Není tedy potřeba řešit incident s duplicitními adresami.

2.8 Automatizace

K automatizaci je využit automatizační nástroj Ansible a šablonový jazyk jinja2. Taktéž je využito orchestrační služby Heat v OpenStacku.

V následujících podkapitolách této kapitoly je popsán postup automatizace Jump serveru a jeho funkcí. Na začátku bylo nutné zautomatizovat tvorbu serveru s Jump sítí. Následně se do Jump sítě připojuje instance, bez možnosti přidělení plovoucí adresy, v opačném případě musí být přidělena. K instancím připojeným do Jump sítě vzniká SSH tunel, jenž se tvoří mezi Jump serverem a instancí. Tento tunel mohou využívat studenti pro připojení ke vzdáleným instancím přes Jump server.

Celý scénář je automatizován podle předvytvořeného konfiguračního souboru, který definuje proměnné pro daný scénář, popřípadě počet vytvořených scénářů, jenž určuje počet studentů, kteří mají scénář využít. Tento konfigurační soubor s proměnnými navrhl autor, který zautomatizoval vytváření scénářů Kybernetické arény, dle počtu studentů.

2.8.1 Jump server a Jump síť

Pro automatizaci Jump serveru a Jump sítě byla vytvořena šablona Heatu. V šabloně jsou nadefinované proměnné pro server i síť, jenž po použití orchestrace vytvoří topologii Jump serveru, která byla předtím vytvořena manuálně. Příklad struktury vytvořené šablony heatu Jump serveru:

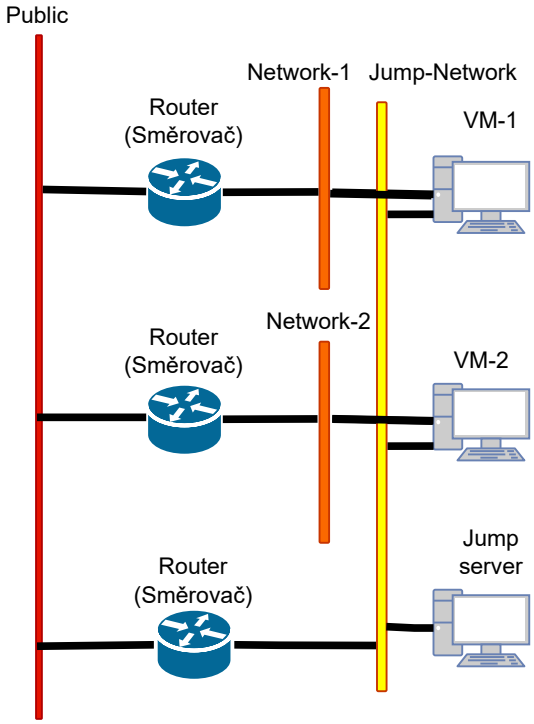
Výpis 2.1: Příklad šablony heat pro Jump server.

```
heat_template_version: 2018-08-31

# Network-JumpNetwork
resources:
  JumpNetwork:
    type: OS::Neutron::Net
    properties:
      name: {{ jump.network.name }}
  JumpSubnet:
    type: OS::Neutron::Subnet
    depends_on: [ JumpNetwork ]
    properties:
      name: {{ jump.network.subnet }}
      network_id: { get_resource: JumpNetwork }
      ip_version: 4
      cidr: "{{ jump.network.cidr }}"
      gateway_ip: "{{ jump.network.gateway_ip }}"

## Network-public
  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: "{{ jump.network.external_gateway }}"
## VM
  JumpServerPort:
    type: OS::Neutron::Port
    depends_on: [ JumpSubnet ]
    properties:
      network: { get_resource: JumpNetwork }
  JumpServer:
    type: OS::Nova::Server
    depends_on: [ JumpServerPort ]
    properties:
      name: {{ jump.server.name }}
      flavor: {{ jump.server.flavor }}
      image: {{ jump.server.image }}
      networks:
        - port: { get_resource: JumpServerPort }
      user_data_format: RAW
      user_data: |
        #cloud-config
        runcmd:
          - echo "Host *" > /home/{{ user_name }}/.ssh/config
          - echo " StrictHostKeyChecking no" >> /home/{{ user_name }}/.ssh/config
        ssh_pwauth: true
        users:
          - name: {{user_name}}
            groups: sudo
            sudo: ['ALL=(ALL) NOPASSWD:ALL']
            lock_passwd: false
            shell: /bin/bash
            passwd: "{{ user_password }}"
            chpasswd: {expire: False}
            ssh-authorized-keys:
```

Pro automatizaci musel být scénář pro Jump server rozšířený o směrovač, který je připojený k public síti. K směrovači je připojena Jump síť, v níž je Jump server umístěn 2.11.



Obr. 2.11: Topologie s umístěným Jump serverem.

Orchestrace dle šablony probíhala za pomocí automatizačního nástroje Ansible. Autor jenž navrhl automatizaci pro scénáře vytvořil adresář game-orchestrator, která obsahuje konfigurační soubory typu yalm pro automatizaci přes nástroj Ansible. V tomto souboru byla provedena konfigurace automatizace v rámci zadání této práce.

Pro automatizaci s využitím vytvořené šablony Heatu byla vytvořena nová role, která se skládá ze souborů, v nichž jsou konfigurační soubory typu yalm, v kterých je specifický zdrojový kód pro automatizaci dat a skripty pro danou automatizační úlohu. Role je alokována v tzv. playbooku. Playbook je soubor, který mapuje cestu k rolím s úkoly.

Adresář role pro tvorbu Jump serveru byl pojmenován template-jumpserver, jenž se skládá z dalších tří adresářů `defaults`, `tasks` a `templates`. V prvním adresáři byla implementována funkce pro tvorbu adresáře Jump serveru na straně kontroléru Openstacku. Adresář `tasks` obsahuje soubor s funkcemi pro vytvoření Jump serveru

z již zmíněné šablony serveru. Příklad adresáře `tasks` s funkcemi v souboru main pro automatizaci Jump serveru:

Výpis 2.2: Příklad adresáře tasks s funkcemi v souboru main pro automatizaci Jump serveru.

```
- name: Generate game template for Jump Server
  template:
    src: jump_template.yml
    dest: "{{ game_config_dir }}"
    force: yes
    mode: 0664

- name: Validate template
  command: "{{ openstack_project_command }} orchestration template validate --template {{
    ↪heat_config_path }}"
  changed_when: False

- name: Get existing stacks
  command: "{{ openstack_project_command }} stack list -f value -c 'Stack Name'"
  changed_when: False
  register: existing_stacks

- name: Create stack from generated template
  command: "{{ openstack_project_command }} stack create --timeout 20 --template {{
    ↪heat_config_path }} {{ jump.stack.name }}"
  changed_when: True
```

2.8.2 Přidělování IP adres, nebo připojení k Jump síti

Pro připojení instancí k Jump síti, bylo nutné upravit hlavní skript pro tvorbu scénářů. Úprava proběhla podle hlavního konfiguračního souboru `config.yml`, v kterém byla stanovena podmínka: V případě možnosti "ssh connection : no"bylo nutné zajistit připojení instance k Jump síti. U druhé podmínky při "ssh connection : yes"bylo nutné zajistit přidělení plovoucí IP adresy vytvořené instanci. Pro splnění podmínek byla přidána konfigurace do yaml souboru pro vytváření scénářů. Po připsání kódu se vytváří nový scénář podle daných podmínek. Instance ke kterým je možné přidělit plovoucí IP adresu ji obdrží, a v druhém případě je instance připojena k Jump síti.

Výpis 2.3: Přidaný kód dle podmínek ssh konektivity pro scénář.

```
{% for server in ex_vars.vms %}
{%- set server_loop = loop %}
{%- set server_name = server.name|default('server' ~ server_loop.index ) %}
{%- for server_network in server.networks %}
{% if server.ssh_connection %}
  {{ server_name }}-floating-ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: {{jump.network.external_gateway}}
  associate-{{ server_name }}-floating-ip:
    type: OS::Neutron::FloatingIPAssociation
    depends_on: {{ server_name }}
    properties:
      floatingip_id: { get_resource: {{ server_name }}-floating-ip }
      port_id: { get_resource: {{ server_name }}-port{{ loop.index }} }

{% elif not server.ssh_connection %}
  {{ server_name }}-js-port:
    type: OS::Neutron::Port
    properties:
      network: {{ jump.network.name }}
{% endif %}
{% endfor %}
{% endfor %}
```

Taktéž pro instance byl vytvořen unikátní uživatel s heslem pro připojení přes plovoucí IP adresu, popřípadě přes Jump server. Je možné v konfiguračním souboru přidávat veřejné ssh klíče, jenž umožní připojit se bez zadání hesla. V automatizaci byl klíč využíván pro ladění, a hlavně je zadáván do Jump serveru pro přístup administrátora, který posílá příkazy do Jump serveru. Též je nutné přidat předem generovaný RSA klíč pro Jump server, který je použit k tvorbě SSH tunelu. Díky vlastnosti klíče není potřeba zadávat heslo při SSH spojení k VM.

Výpis 2.4: Připojení portů dle podmínek a tvorba uživatele.

```
{% for server_network in server.networks %}
       - port: { get_resource: {{ server_name }}-port{{ loop.index }} }
{% endfor %}
{% if not server.ssh_connection %}
       - port: { get_resource: {{ server_name }}-js-port }
{% endif %}
     user_data_format: RAW
     user_data: |
       #cloud-config
       ssh_pwauth: true
       users:
         - name: {{user_name}}
           groups: sudo
           sudo: ['ALL=(ALL) NOPASSWD:ALL']
           lock_passwd: false
           shell: /bin/bash
           passwd: "{{ user_password }}"
           chpasswd: {expire: False}
           ssh-authorized-keys:
{% for key in keys %}
             - {{key}}
{% endfor %}
{% endfor %}
```

2.8.3 Nastavení SSH tunelu

Pro sestavení SSH tunelu mezi Jump serverem a instancemi připojenými do Jump sítě, byly vytvořeny dva adresáře rolí. První adresář `ssh_tunel` obsahuje tvorbu šablon pro jednotlivý scénář studenta. Tvorba šablon je napsána jazykem jinja2, který se specifikuje tvorbu šablon pro automatizaci. Tento soubor je umístěn v adresáři `templates`.

Pro automatizaci byl lehce pozměněn SSH příkaz pro tvorbu tunelu. Aby bylo umožněno SSH tunel vytvořit na Jump serveru první adresa v příkazu je 0.0.0.0 místo jeho plovoucí adresy. Při zadání 0.0.0.0 umožníme serveru dle nastavení odposlouchávat, popřípadě přeposílat na všech jeho adresách. Na tomto případě server přeposílá studenta na všech jeho adresách a určitém portu k instanci, avšak u studenta se nic nemění, stále zadává port a plovoucí adresu Jump serveru s uživatelem instance.

Výpis 2.5: Ukázka jinja2 šablony pro generování šablon k scénářům pro SSH tunel.

```
{% for vm in awx_extra_vars.vms %}
{% set idx = loop.index - 1 %}
{% if not vm.ssh_connection %}
- block:
    - name: Create {{ vm.name }} SSH tunnel
      shell: "ssh -fNL 0.0.0.0:{{ '{{' }} port{{ idx }} {{ '}}' }}:{{ '{{' }} server{{ idx
        ↪}}.attributes.addresses.{{jump.network.name}}[0].addr {{ '}}' }}:22 {{user_name}}@{{ '{{'
        ↪}} server{{ idx }}.attributes.addresses.{{jump.network.name}}[0].addr {{ '}}' }}"
    - name: Add crontab entry
      cron:
        reboot: true
        job: "ssh -fNL 0.0.0.0:{{ '{{' }} port{{ idx }} {{ '}}' }}:{{ '{{' }} server{{ idx
          ↪}}.attributes.addresses.{{jump.network.name}}[0].addr {{ '}}' }}:22 {{user_name}}@{{
          ↪'{{' }} server{{ idx }}.attributes.addresses.{{jump.network.name}}[0].addr {{ '}}' }}"
  delegate_to: {{ '"{{' }} JSdata.stdout {{ '}}"' }}
{% endif %}
{% endfor %}

- name: Dump connection details for {{ '"{{' }} stack_name {{ '}}"' }}
  copy:
    dest: {{game_config_dir}}/{{ '{{' }} stack_name {{ '}}' }}
    content: |
{% for vm in awx_extra_vars.vms %}
{% set idx = loop.index - 1 %}
{% if vm.ssh_connection %}
      - {{ vm.name }} ==> user={{user_name}}, password={{vm_password}}, ip_address={{ '"{{' }}
        ↪server{{idx}}.attributes.addresses {{ '}}"' }}
{% elif not vm.ssh_connection %}
      - {{ vm.name }} ==> user={{user_name}}, password={{vm_password}}, ip_address={{ '"{{' }}
        ↪server{{idx}}.attributes.addresses {{ '}}"' }}, port={{ '"{{' }} port{{idx}} {{ '}}"' }}
{% endif %}
{% endfor %}
```

V druhém adresáři `tasks` je vytvořena úloha pro ukládání šablon vytvořených generátorem do určitého adresáře.

Výpis 2.6: zvolený adresář pro ukládání vygenerovaných scénářů.

```
---
- template:
    src: mytemplate.yml.j2
    dest: /home/student/game-orchestrator-sshtunel/roles/post-deploy/tasks/{{ stack_name}}.yml
  delegate_to: localhost
```

Poslední vytvořený adresář v rolích byl pojmenován `post-deploy`. Právě do jeho adresáře se vytváří vygenerované šablony pro SSH tunel. Vytvořená šablona podle přidělené plovoucí adresy pro Jump server vytváří v Jump serveru SSH tunel k instancím, jenž to mají v podmínce. Je zde i vytvořený adresář, kde je předvytvořen zmíněný RSA klíč pro Jump server.

Výpis 2.7: Ukázka vygenerované šablony pro vytvoření SSH dle pravidel.

```
- block:
    - name: Create server3 SSH tunnel
      shell: "ssh -fNL 0.0.0.0:{{ port2 }}:{{ server2.attributes.addresses.jump_network[0].addr
        ↪}}:22 student@{{ server2.attributes.addresses.jump_network[0].addr }}"
    - name: Add crontab entry
      cron:
        reboot: true
        job: "ssh -fNL 0.0.0.0:{{ port2 }}:{{ server2.attributes.addresses.jump_network[0].addr
          ↪}}:22 student@{{ server2.attributes.addresses.jump_network[0].addr }}"
  delegate_to: "{{ JSdata.stdout }}"

- name: Dump connection details for "{{ stack_name }}"
  copy:
    dest: /home/student/game_configurations/testtunel/{{ stack_name }}
    content: |
      - server1 ==> user=student, password=12345, ip_address="{{ server0.attributes.addresses }}"
      - server2 ==> user=student, password=12345, ip_address="{{ server1.attributes.addresses }}"
      - server3 ==> user=student, password=12345, ip_address="{{ server2.attributes.addresses }}",
        ↪port="{{ port2 }}"
```

2.9 Testování automatizace

V této kapitole je popsána funkčnost automatizace Jump serveru se scénáři. Nedílnou částí je konfigurační soubor ve kterém jsou alokovány proměnné pro scénáře, a i pro samotný Jump server. Ověření proběhlo na třech vygenerovaných scénářích.

2.9.1 Konfigurace

Pro ověření automatizace byl nejprve upraven konfigurační soubor `config.yml`. Při automatizaci Jump serveru byly jeho proměnné nastavovány zde, taktéž i pro vytvořené scénáře. Je zde nakonfigurováno s jakými proměnými bude scénář sestaven. Například jaký bude obraz, flavor, název stacku, název instancí, serveru apod.

Výpis 2.8: Konfigurace proměnných pro Jump server.

```
jump:
  network:
    name: jump_network
    subnet: jump_subnet
    cidr: 192.168.100.0/24
    gateway_ip: 192.168.100.1
    external_gateway: public
    router: Jump-Router
  server:
    name: Jump-Server
    flavor: arena.1-1-5
    image: debian-10
  stack:
    name: Jump-Stack
```

Výpis 2.9: Příklad konfigurace pro instanci server1.

```
game_name: testtunel
user_name: student
vm_password: 12345
keys:

student_count: 3
awx_extra_vars:
  vms:
    - name: server1
      image: ubuntu_minimal
      flavor: arena.1-1-5
      networks:
        - ['net1', '192.168.10.11']
      console: yes
      ssh_connection: yes
```

U výpisu pro server1 je taktéž vyobrazeno nastavení jména a hesla pro uživatele a také název celého scénáře, jenž byl pojmenován testtunel. U proměnné keys je nutné zadat klíče administrátorů, který spouští daný scénář a zároveň bylo nutné předgenerovat veřejný RSA klíč Jump server, který jej používá pro tvorbu SSH

tunelu k instancím bez možnosti použití hesla. Tento klíč s veřejným klíčem byl předem vygenerován v adresáři role `post-deploy/files`.

2.9.2 Spuštění automatizace

Spuštění celé vytvořené automatizace probíhal na vzdáleném serveru, na kterém byl předinstalován nástroj ansible. Aby automatizace Jump serveru proběhla musel být na serveru vytvořen stejný uživatel jak na kontroléru OpenStacku, což byl uživatel student. Vytvořené a předvytvořené role s úlohami byly nadefinovány v playbooku `game_start.yml`. Ve výchozí podobě byly zvoleny pro instance obrazy cirros, ale u nich se vyskytoval problém při tvorbě uživatele, a proto byly zvoleny dva typy obrazů od dvou rozdílných distributorů.

Výpis 2.10: Nastavení rolí v hlavním playbooku pro automatizaci.

```
---
# game_id
- hosts: control
  gather_facts: yes
  vars:
    ansible_user: "{{user_name}}"
  tasks:
    - include_role:
        name: preactions
    - include_role:
        name: template-jumpserver
    - include_role:
        name: template-generator
    - include_role:
        name: ssh_tunnel
      vars:
        stack_name: "{{ game_name }}{{ cycle }}"
      with_sequence:
        start=1
        end="{{ student_count }}"
      loop_control:
        loop_var: cycle
    - include_role:
        name: post-deploy
        tasks_from: "{{ game_name }}{{ cycle }}.yml"
      vars:
        stack_name: "{{ game_name }}{{ cycle }}"
      with_sequence:
        start=1
        end="{{ student_count }}"
      loop_control:
        loop_var: cycle
```

Po nastavení adresářů v playbooku je možné jej spustit příkazem:

```
ansible-playbook -i inv.ini game_start.yml -e @config.yml -e ansible_user=student
```

Bylo nutné vytvořit nového uživatele ansible, aby měly potřebné úlohy přístup k nadefinovaným adresářům, také se zadává přednastavené heslo skrz přístup pro vytvořené funkce. Po potvrzení příkazu se podle pořadí rolí spouštěli jednotlivé úlohy v nichž jsou nadefinovány. Po dokončení relací byly vytvořeny scénáře s Jump serverem. Vytvořené instance s Jump serverem je možné vidět na obrázku 2.12.

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status
testtunel3-server3	debian-10	**testtunel3-net2** 192.168.20.13 **jump_network** 192.168.100.9	arena.1-1-5	-	Aktivní
testtunel3-server2	ubuntu_minimal	192.168.10.12, 10.0.3.241	arena.1-1-5	-	Aktivní
testtunel3-server1	ubuntu_minimal	192.168.10.11, 10.0.1.206	arena.1-1-5	-	Aktivní
testtunel2-server3	debian-10	**testtunel2-net2** 192.168.20.13 **jump_network** 192.168.100.108	arena.1-1-5	-	Aktivní
testtunel2-server1	ubuntu_minimal	192.168.10.11, 10.0.3.165	arena.1-1-5	-	Aktivní
testtunel2-server2	ubuntu_minimal	192.168.10.12, 10.0.4.244	arena.1-1-5	-	Aktivní
testtunel1-server2	ubuntu_minimal	192.168.10.12, 10.0.4.64	arena.1-1-5	-	Aktivní
testtunel1-server1	ubuntu_minimal	192.168.10.11, 10.0.1.147	arena.1-1-5	-	Aktivní
testtunel1-server3	debian-10	**testtunel1-net2** 192.168.20.13 **jump_network** 192.168.100.70	arena.1-1-5	-	Aktivní
Jump-Server	debian-10	192.168.100.185, 10.0.4.96	arena.1-1-5	-	Aktivní

Obr. 2.12: Vytvořené scénáře s Jump serverem.

Pro všechny instance jenž měly v podmínce "ssh connection : yes"jim byla přiřazena plovoucí IP adresa. Přes tuto adresu je možné se do instance připojit. Uživatel ve všech instancí je nastaven na student a heslo 12345. Pro instance s podmínkou "ssh connection : yes"byly vygenerovány přes šablonu `mytemplate.yml.j2` vytvořeny dle počtu scénářů šablony. Jelikož byl test prováděn pro tři scénáře, tudíž byly vytvořeny tři šablony 2.14.

```
[student@ansible-jumpserver tasks]$ ls
testtunel1.yml  testtunel2.yml  testtunel3.yml
[student@ansible-jumpserver tasks]$
```

Obr. 2.13: Vytvořené šablony přes .j2 generátor.

V průběhu testování docházelo k problému, jelikož vygenerované šablony pro každý scénář obsahoval příkaz, který potřeboval balíček jq. Balíček by se musel nainstalovat do OpenStacku, k tomu jsou potřeba práva a RHEL subscription. Generovací šablona `mytemlate.yml.j2` byla upravena, aby generovala jiný příkaz pro získání IP adresy.

Výpis 2.11: Nefunkční vygenerovaná šablona s příkazem k získání floating IP Jump serveru.

```
- name: Get Servers Info for "{{ jump.stack.name }}"
shell: "{{ openstack_project_command }} stack resource show Jump-Stack floating_ip -c attributes -f
  ↪value --format json | jq .attributes.floating_ip_address | tr -d '\"' "
register: JSdata
```

Výpis 2.12: Funkční vygenerovaná šablona s příkazem k získání floating IP Jump serveru. (před znakem dolaru bez lomítka)

```
- name: Get Servers Info for "{{ jump.stack.name }}"
  shell: "{{ openstack_project_command }} stack resource show Jump-Stack floating_ip -c attributes
    ↪-f yaml |awk '/floating_ip_address/ {print \$2}'"
  register: JSdata
```

Po upravení příkazu bylo možné pokračovat ve spuštění playbooku od role, u které se vyskytoval problém. Pro generaci šablon k SSH tunelu slouží již zmíněná role `ssh_tunel`. Pro spuštění od dané role byly již proběhlé role zakomentovány. Po spuštění byly cyklicky generovány pro každý stack scénáře nové šablony s již změněným příkazem. Po vygenerování byla spuštěna role `post-deploy`. Tato role cyklicky spouští šablony pro tvorbu SSH tunelu. Nejprve bylo nutné potvrdit zkopírovaní klíče do Jump serveru. Po zkopírování proběhlo na přidělené plovoucí IP adrese Jump serveru konfigurace SSH tunelu k instancím. Porty se nastavují náhodně podle zvolaného rozsahu. Po úspěšném nastavení je konfigurace vytištěna cyklicky do souborů v kontroléru OpenStacku.

```
[student@arena-ctrl01 testtunel]$ ls
heat_template.yml  jump_template.yml  testtunel1  testtunel2  testtunel3
[student@arena-ctrl01 testtunel]$
```

Obr. 2.14: Vypsané soubory s nastavenými konfiguracemi.

V každém souboru `testtunel2`, `testtunel2`, `testtunel3` je vypsána konfigurace. Jsou zde viditelné přidělené plovoucí adresy, a v případě vytvoření SSH tunelu je u instance vypsán port.

Výpis 2.13: Výpis informací pro druhý stack scénáře testtunel2.

```
- server1 ==> user=student, password=12345, ip_address="{'testtunel2-net1': [{'version': 4, 'addr':
  ↪'192.168.10.11', 'OS-EXT-IPS:type': 'fixed', 'OS-EXT-IPS-MAC:mac_addr': 'fa:16:3e:94:84:c7'},
  ↪{'version': 4, 'addr': '10.0.3.165', 'OS-EXT-IPS:type': 'floating', 'OS-EXT-IPS-MAC:mac_addr':
  ↪'fa:16:3e:94:84:c7'}]}"
- server2 ==> user=student, password=12345, ip_address="{'testtunel2-net1': [{'version': 4, 'addr':
  ↪'192.168.10.12', 'OS-EXT-IPS:type': 'fixed', 'OS-EXT-IPS-MAC:mac_addr': 'fa:16:3e:e3:6c:bc'},
  ↪{'version': 4, 'addr': '10.0.4.244', 'OS-EXT-IPS:type': 'floating', 'OS-EXT-IPS-MAC:mac_addr':
  ↪'fa:16:3e:e3:6c:bc'}]}"
- server3 ==> user=student, password=12345, ip_address="{'testtunel2-net2': [{'version': 4, 'addr':
  ↪'192.168.20.13', 'OS-EXT-IPS:type': 'fixed', 'OS-EXT-IPS-MAC:mac_addr': 'fa:16:3e:2b:ee:2b'}],
  ↪'jump_network': [{'version': 4, 'addr': '192.168.100.108', 'OS-EXT-IPS:type': 'fixed',
  ↪'OS-EXT-IPS-MAC:mac_addr': 'fa:16:3e:0d:5c:8e'}]}", port="20337"
```

V scénáři testtunel2 byl pro VM server3 zvolen náhodný port 20337. Pokud student použije příkaz:

```
ssh -p20337 student@10.0.4.96
```

Adresa 10.0.4.96 je přidělená plouvoucí adresa Jump serveru. Pro kontrolu je možné použít panel v OpenStacku 2.12. U VM server1 a server2 se shoduje přidělená plovoucí adresa. Celý scénář je možné vidět v tabulce 2.7.

testtunel1	
server1	IP : 10.0.1.147
server2	IP : 10.0.4.64
server3	PORT : 15516
testtunel2	
server1	IP : 10.0.3.165
server2	IP : 10.0.4.244
server3	PORT : 20337
testtunel3	
server1	IP : 10.0.1.206
server2	IP : 10.0.3.241
server3	PORT : 3226
Jump server	
IP : 10.0.4.96	

Tab. 2.7: Informace o scénářích s Jump serverem.

2.9.3 Výstup testování automatizace

Dle dosažených výsledků v rámci testu automatizace bylo vyhodnoceno za zdařilé. VM, k nimž je možné přidělit plouvoucí IP adresu, ji obdrželi, a v druhém případě byl k instancím vytvořen funkční SSH tunel. Připojení k instancím přes přidělený

port funguje, a na obrázku 2.15 je možné vidět úspěšné připojení přes Jump server k instanci ve scénáři testtunel3.

```
xkomar31@MSI:~$ ssh -p3226 student@10.0.4.96
The authenticity of host '[10.0.4.96]:3226 ([10.0.4.96]:3226)' can't be established.
ECDSA key fingerprint is SHA256:EcgZ3lWdU1GDRCroaR0uSbphf1IdhCa4k6QC5YqDkp4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.0.4.96]:3226' (ECDSA) to the list of known hosts.
student@10.0.4.96's password:
Linux testtunel3-server3 4.19.0-18-cloud-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
student@testtunel3-server3:~$
```

Obr. 2.15: Ukázka připojení do serveru3 ve scénáři testtunel1.

Pro připojení k instanci za využití plovoucí adresy byla konektivita odzkoušena k instanci v scénáři testtunel2. Jedná se o VM server2 s přidělenou IP adresou 10.0.4.244 2.16.

```
xkomar31@MSI:~$ ssh student@10.0.4.244
The authenticity of host '10.0.4.244 (10.0.4.244)' can't be established.
ECDSA key fingerprint is SHA256:mzb6aIdDQPkoCulK82wHVIZMBXXDMvIT0U86uLNLtO8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.4.244' (ECDSA) to the list of known hosts.
student@10.0.4.244's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1063-kvm x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

0 packages can be updated.
0 updates are security updates.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

student@testtunel2-server2:~$
```

Obr. 2.16: Ukázka připojení do serveru2 ve scénáři testtunel2.

2.9.4 Konfigurace Jump serveru

Pro testování automatizace nebyl dle stanovených podmínek Jump server nakonfigurován. Je nutné zajistit zabezpečení, aby do něj nikdo neměl přístup s výjimkou administrátora, který k němu musí mít přístup přes klíč, jelikož přístup využívá pro samotnou konfiguraci SSH tunelů. Pro zabezpečení Jump serveru je připsán v `config.yml` uživatel a heslo pro jump server. Nastavení přepíše výchozího uživatele i s heslem.

Výpis 2.14: Přepsání výchozího uživatele s nastavenými proměnnými.

```
#cloud-config
runcmd:
- echo "Host *" > /home/{{ user_name }}/.ssh/config
- echo " StrictHostKeyChecking no" >> /home/{{ user_name }}/.ssh/config
ssh_pwauth: true
 user:
 name: {{user_jump}}
 groups: sudo
 sudo: ['ALL=(ALL) NOPASSWD:ALL']
 lock_passwd: false
 shell: /bin/bash
 passwd: "{{ jump_password }}"
 chpasswd: {expire: False}
 ssh-authorized-keys:
```

Závěr

Hlavním cílem práce bylo navrhnout Jump server pro SSH tunelování k instancím na platformě OpenStack. Práce v první kapitole seznamuje čtenáře s teorií problematiky. Zejména jsou zde vysvětleny pojmy ohledně virtualizace a kontejnerizace a jak je možné je využít v praxi. Následně je popsána samotná platforma OpenStack s jejími službami. V rámci jednotlivých služeb jsou popsány postupy k jejich využití. Například jak vytvořit instanci, síť. Na závěr teoretické části je porovnání VMs a kontejnerů a na základě tohoto porovnání je z počátku pro Jump server vybrán kontejner.

V praktické části je manuální implementace Jump serveru jako kontejner. Před samotnou implementací je nejprve vytvořen obraz, který je importován do kontejneru v OpenStacku. Následně je využit pro testování SSH konektivity. Z výsledku testování byl na straně kontejneru problém při získávání výchozí brány sítě. V kontejneru nelze měnit směrování. Vzhledem k této chybě byl Jump server implementován instance (VM). Na základě chyby v kontejneru probíhalo v praktické části manuální ověření Jump serveru jako VM. Ověření proběhlo úspěšně a musel být server automatizován pro scénáře v Kybernetické aréně v OpenStacku.

V případě vytváření a realizace bylo nutné, se nejprve seznámit a naučit pracovat v platformě OpenStack. Bylo nutné nastudovat problematiku kontejnerizace a hlavně samotných kontejnerů. Na základě získaní znalostí bylo možné vytvořit vlastní obraz pro kontejner. Avšak v případě ověření funkčnosti nastával již zmíněný problém.

Pro automatizaci Jump serveru se bylo nutné především naučit v automatizačním nástroji Ansible, a také se naučit syntax jazyka jinja2, jenž se používá pro tvorbu šablon v Ansible. Vzhledem k tomu, že autor práce byl bez zkušeností s těmito automatizačními nástroji, byla práce o něco složitější.

V poslední praktické části probíhá automatizace Jump serveru. Je zde popsán postup, jak autor práce postupoval v automatizaci, co bylo nutné vytvořit a jakým způsobem vypadá kód automatizace. Na závěr probíhá ověření automatizace, jenž je prováděno na třech scénářích.

Literatura

[1] History of Virtualisation. *Probrand* [online]. Birmingham: Probrand [cit. 2021-10-30]. Dostupné z: <https://www.probrand.co.uk/it-services/vmware-solutions/history-of-virtualisation>

[2] DAKIC, V.; CHIRAMMAL, H. D.; MUKHEDKAR, P.; VETTATHU, A. *Mastering KVM Virtualization* [online]. Second edition. Packt Publishing, October 2020 [cit. 2021-10-30]. ISBN 9781838828714. Dostupné z: <https://www.oreilly.com/library/view/mastering-kvm-virtualization/9781838828714/>

[3] What is virtualization? *Red Hat* [online]. Red Hat, © 2021, March 2, 2018 [cit. 2021-10-30]. Dostupné z: <https://www.redhat.com/en/topics/virtualization/what-is-virtualization#types-of-virtualization>

[4] Co je virtualizace? *Cs education-wiki* [online]. @2021 [cit. 2021-10-30]. Dostupné z: <https://cs.education-wiki.com/2600022-what-is-virtualization>

[5] Virtualization. *IBM* [online]. IBM Cloud Education, 19 June 2019 [cit. 2021-10-30]. Dostupné z: <https://www.ibm.com/cloud/learn/virtualization-a-complete-guide>

[6] What is virtualization? *Open Source* [online]. Red Hat, ©2021 [cit. 2021-10-30]. Dostupné z: <https://opensource.com/resources/virtualization>

[7] STODŮLKA, Tomáš. *Platforma pro virtualizaci komunikační infrastruktury*. Brno, 2020. Dostupné z: <https://www.vut.cz/studenti/zav-prace/detail/125999> Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Václav Uher, PhD.

[8] *Domovská stránka libvirt projektu* [online]. [cit. 2021-12-10]. Dostupné z: <https://libvirt.org/>

[9] Containerization. *IBM* [online]. IBM Cloud Education, 23 June 2021 [cit. 2021-10-30]. Dostupné z: <https://www.ibm.com/cloud/learn/containerization>

[10] POLEDNIK, Martin. *Virtualizace a kontejnerizace* [online]. [cit. 2021-10-30]. Dostupné z: <https://www.fi.muni.cz/~kas/pv090/referaty/2014-podzim/virt.html>

[11] SCHENKER, Gabriel N., Hideto SAITO, Hui-Chuan Chloe LEE a Ke-Jou Carol HSU. *Getting Started with Containerization* [online]. March 2019. Packt Publishing [cit. 2021-10-30]. ISBN 9781838645700. Dostupné z: <https://www.oreilly.com/library/view/getting-started-with/9781838645700/>

[12] *Container Runtime* [online]. Insu Jang, © 2017 - 2021, Oct 31, 2019 [cit. 2021-10-30]. Dostupné z: <https://insujang.github.io/2019-10-31/container-runtime/>

[13] *Managing Kubernetes: operating Kubernetes clusters in the real world* [online]. Beijing: O'Reilly Media, Incorporated, 2018 [cit. 2021-10-30]. ISBN 978-1-492-03391-2. Dostupné z: <https://1url.cz/aKDMH>

[14] What are containers? *NetApp* [online]. NetApp, © 2021 [cit. 2021-12-10]. Dostupné z: <https://www.netapp.com/devops-solutions/what-are-containers/>

[15] DOCKER. Cgroups, namespaces, and beyond: what are containers made from? *YouTube* [online]. YouTube video, 2015 [cit. 2021-10-30]. Dostupné z: <https://www.youtube.com/watch?app=desktop&v=sK5i-N34im8&ab_channel=Docker>

[16] IBM Cloud Education. Docker. *IBM* [online]. 23 June 2021 [cit. 2021-12-10]. Dostupné z: <https://www.ibm.com/cloud/learn/docker>

[17] MIŠUREC, Jiří. *Kybernetická aréna pro výzkum, testování a edukaci v oblasti kyberbezpečnosti*. Brno, 01.07.2019 - 30.06.2022n. l. Dostupné z: <https://www.vut.cz/vav/projekty/detail/30291>. Projekt. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[18] CHAPTER 9. CONNECT AN INSTANCE TO THE PHYSICAL NETWORK. *Red Hat: Customer Portal* [online]. Red Hat, © 2021 [cit. 2021-12-10]. Dostupné z: <https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/8/html/networking_guide/sec-connect-instance>

[19] OpenStack Compute (nova). *Openstack* [online]. Rackspace Cloud Computing [cit. 2021-12-10]. Dostupné z: https://docs.openstack.org/nova/latest/#architecture-overview/>

[20] Overview. *Openstack* [online]. Rackspace Cloud Computing [cit. 2021-12-10]. Dostupné z: <https://docs.openstack.org/zun/latest/install/overview.html>

[21] Welcome to the Heat documentation! *Openstack* [online]. Rackspace Cloud Computing [cit. 2021-12-10]. Dostupné z: <https://docs.openstack.org/heat/latest/>

[22] KUMAR SINGH, Pradeep a Madhuri KUMARI. *Containers in OpenStack*. Packt Publishing, December 2017. ISBN 9781788394383.

[23] Runtime options with Memory, CPUs, and GPUs. *Docker Docs* [online]. Docker, © 2013-2021 [cit. 2021-12-10]. Dostupné z: <https://docs.docker.com/config/containers/resource_constraints/>

[24] 3 Types of Container Runtime and the Kubernetes Connection. *Aqua Security* [online]. Aqua Security Software, © 2021 [cit. 2021-12-10]. Dostupné z: <https://www.aquasec.com/cloud-native-academy/container-security/container-runtime/>

Seznam symbolů a zkratek

API	Application Programming Interface
DHCP	Dynamic Host Configuration Protocol
HW	Hardware
KVM	Kernel-based Virtual Machine
NAT	Network Address Translation
OS	Operating System
QEMU	Quick EMUlator
SSH	Secure Shell
SW	Software
VM	Virtual Machine
WSL	Windows Subsystem for Linux
YAML	Ain't Markup Language