
# 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 2023

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:*** 2022/23

**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:*** ***Termín odevzdání:*** 17.8.2023

***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 z fyzické sítě. Závěrečná práce je složena ze tří částí – teoretické a praktické která se člení na dvě části. 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 Jump server v druhé a třetí části práce. Po manuálním ověření funkčnosti Jump serveru bylo jeho vytváření automatizováno na platformu Openstack.



## KLÍČOVÁ SLOVA

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



## 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 three parts - theoretical and practical, which has two parts. 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 second and third 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, Ansible

KOMÁREK, Ladislav. *Příprava a implementace Jump serveru pro herní scénáře v Kybernetické aréně*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 64 s. Bakalářská práce. Vedoucí práce: Ing. Tomáš Stodůlka

# Prohlášení autora o původnosti díla

| | |
|---|---|
| **Jméno a příjmení autora:** | Ladislav Komárek |
| **VUT ID autora:** | 211798 |
| **Typ práce:** | Bakalářská práce |
| **Akademický rok:** | 2022/23 |
| **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. A také děkuji bývalé spolužačce za případnou konzultaci při korektuře.

# Obsah

# Seznam obrázků

# Seznam tabulek

# 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 z fyzické sítě 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 tři 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.

Zbylé dvě části tvoří praktickou část bakalářské práce. V první části je popsána manuální implementace Jump serveru a technologií. 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.

Vzhledem k výsledkům z manuální implementaci je v druhé části je prováděna automatizace Jump serveru s technologií virtualizace. Jump server je automatizován se scénáři Kybernetické arény a umožňuje připojení z fyzické sítě do virtuálních instancí. 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

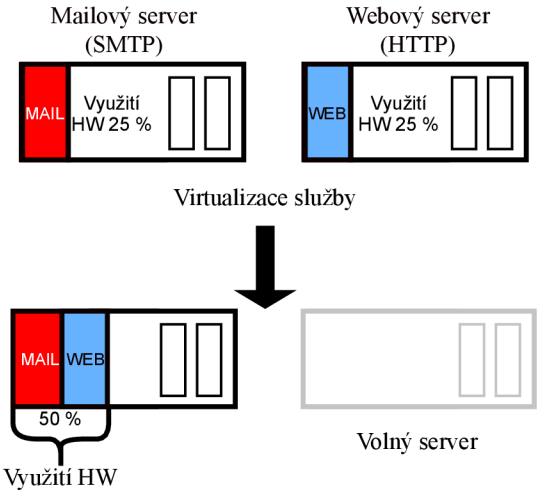
První vývoj virtualizace vznikal od 70. let 20. století firmou IBM za účelem plného využití počítačových zdrojů. Firma tak dokázala redukovat náklady za výpočetní kapacity a plně využít hardware počítače. Později v 90. letech firma VMware vyřešila možnost virtualizace pro hardwarovou počítačovou platformu 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.

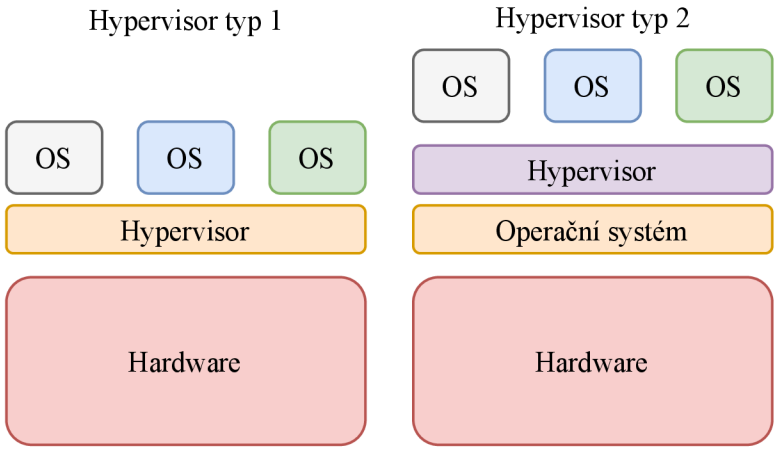
- **Záloha a testování** – VMs je možné využít v rámci testování nových aplikací, jelikož umožňují vytváření záloh. Uživatel nad těmito vytvořenými zálohami má systematický přehled a v případě poškození OS na VM, je možné obnovit původní stav VM ze zálohy. V případě poškození OS na VM hostitelský OS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

### 1.1.3 Ukázka nástroje Libvirt

K vytvoření VMs je možné využít předem vytvořený obraz VM ve formátu qcow2[4]. 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[5].

```
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
```

[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.

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

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

Pro potřeby podrobnějšího výpisu informací o určité VM je příkaz:

```
virsh dominfo debianVM
```

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
```

## 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]:

- **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 souborem pro kontejner je DockerFile (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.

### 1.2.3 Ukázka nástroje Docker

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.

## 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

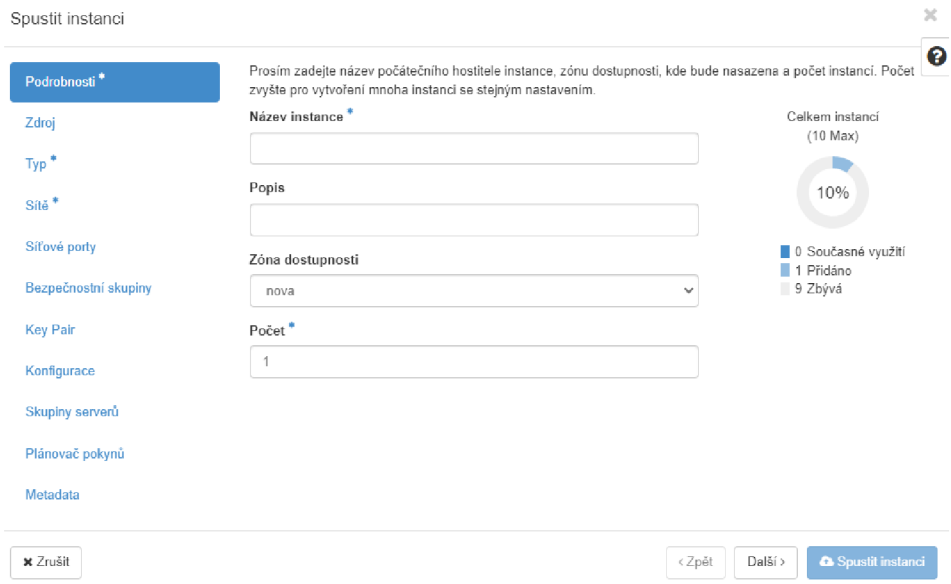
Nova projekt je služba, která podporuje vytváření a správu VMs. V případě Neutronu, kde služba slouží především pro vytváření a správu sítí, tak služba Nova slouží pro výpočetní služby OpenStack. V rámci výpočetní služby nova poskytuje daemony[6] pojmenované `nova-*`. Příkladem deamonu je `nova-compute` 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.
- **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.

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

- **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:

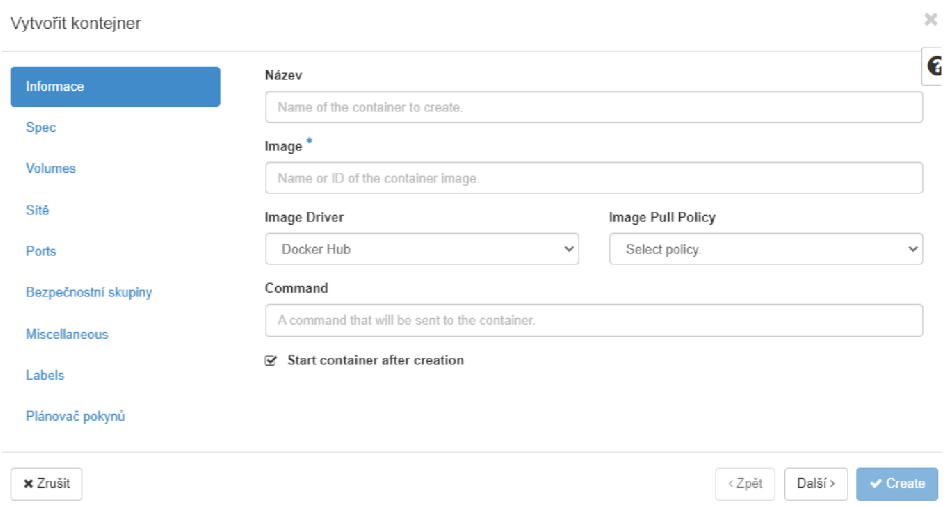
- **Podrobnosti** – je důležité zadat název instance a počet instancí, které budou tvořeny se stejným nastavením.
- **Zdroj** – slouží pro výběr z jakého zdroje bude tvořena instance, je zde na výběr ze čtyř možností: obrazu, snímku obrazu, svazku nebo snímku svazku. Následně je nutné přidělit zvolený 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.



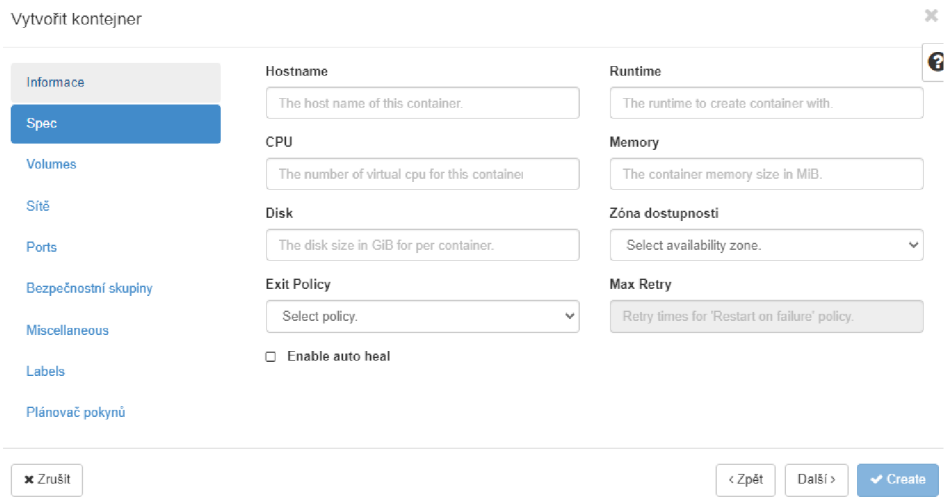

Obr. 1.4: Formulář k vytváření kontejnerů

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[7]. 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.
- Schopnost poskytovat služby pro propojení mezi fyzickou a virtuální sítí, například přes protokol SSH.
- Umožňovat snadnou implementaci na fyzickou síť.
- Integrace s existujícími šablony Heatu.

[7] YALM je typ jazyka pro serializaci dat.

### 1.4.1 Výpočetní kapacity

Hlavní prioritou je minimalizace využití výpočetní kapacity v OpenStacku pro provoz Jump serveru. Z tohoto důvodu je nezbytné provést srovnání těchto dvou odlišných technologií.

- **Kontejner** – Kontejnery sdílí jádro operačního systému hostitele s knihovnami, což snižuje režii kontejneru a také je umožněno rychlejší spouštění a zastavení. Díky sdílení jádra a systémových knihoven jsou kontejnery efektivnější z hlediska paměťového a diskového prostoru a tím kontejnery nepotřebují tolik místa na disku jak VMs. Také dokáže efektivné sdílet ostatní výpočetní prostředky jako je využití procesoru. Z důvodu sdíleného jádra jsou kontejnery omezeny u výběru operačních systémů. Kontejner musí mít operační systém kompatibilní s hostitelským.
- **Virtuální stroj** – Každý virtuální stroj potřebuje svůj vlastní operační systém se všemi knihovnami a zdrojovými soubory, což zapříčiňuje vysoké využiti místa na disku. Jelikož se jedná o virtualizaci, kde VM je plně izolována, a obsahuje celý operační systém, je nutné jí alokovat podstatě větší množství HW zdrojů jak kontejneru pro její provoz.

### 1.4.2 Provoz služeb

Služba SSH není nijak náročná na provoz a je možné ji provozovat na jakékoliv operačním systému. Důležitá je rychlá odezva a rychlá implementace samotné aplikace co se týče do scénářů v Kybernetické aréně.

- **Kontejner** – Kontejnery jsou z hlediska nasazování aplikačních služeb omezeny, jelikož je nutné pouze implementovat aplikace kompatibilní s OS hostitele. Na linuxový operační systém není možné nasadit kontejnerizovanou aplikaci pro OS Windows. Spuštění aplikace přes kontejner je i o několik desítek sekund rychlejší než u VMs, Jelikož není potřeba spouštět celý operační systém. Kontejnery lze i rychle spravovat, replikovat pomocí funkcí kontejnerizace. Také aktualizace a nasazení je mnohem rychlejší vzhledem k VMs.
- **Virtuální stroj** – Virtuální stroje nejsou nijak omezovány co se týče provozu služeb. Pokud si aplikace vyžaduje specifický OS, tak v případě VM je možné zvolit takový který je kompatibilní s aplikační službou. Aplikace jsou u virtualizace plně izolovány, což znamená, že selhání jedné nemusí mít vliv na druhou. Také se to týká v případě poškození nějakých knihoven či souborů OS hostitele, kde u kontejnerizace může dojít k selhání celého kontejneru, a tím i služby.

### 1.4.3 Bezpečnost technologií

Každá z těchto technologii má jiné způsoby zabezpečení. I když s velkou pravděpodobností naše implementovaná služba nebude potřebovat extrémní zabezpečení, tak v případě Kybernetické arény, je dobré vědět odlišnosti zabezpečení těchto technologií. Je možné, že by v budoucnu v rámci testování a i zlepšení mohlo dojít k změnám a různým zkouškám zabezpečení.

- **Kontejner** – Kontejnery nejsou tak izolovány oproti VMs, jelikož sdílejí jádro OS s hostitelem a systémové knihovný, tak je zvýšené riziko šíření útoků mezi kontejnery. Je pravděpodobné, že zranitelnost jednoho kontejneru bude možné využít i na druhý kontejner. Vážné poškození OS hostitele způsobí výpadek kontejnerů, také by mohlo být pravděpodobnější využít zranitelnosti se dostat z kontejneru na OS hostitele.
- **Virtuální stroj** – Virtuální stroje jsou plně izolovány oproti kontejnerům. Pomocí hypervisoru běží na OS hostitele nebo přímo na HW. VM je tedy od OS hostitele izolována a tak i VMs mezi sebou jsou vzájemně izolovány. Poškození jedné VM nemusí mít žádný vliv na chod dalších, popřípadě napadení z VM do druhé je taktéž mnohem nižší než při kontejnerů.

### 1.4.4 Vyhodnocení porovnání

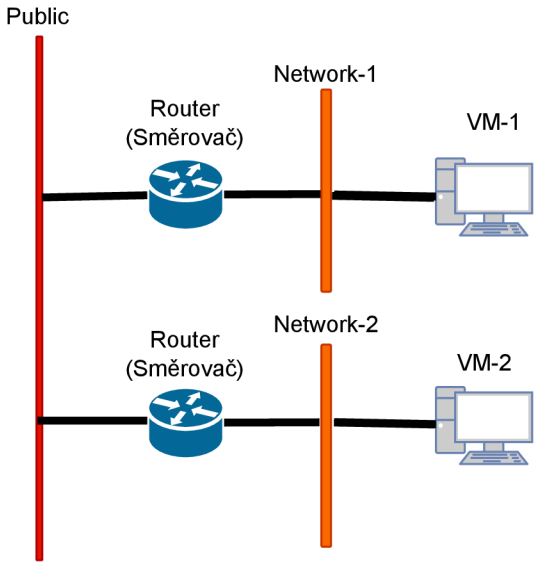
Vzhledem k porovnání je zřejmé, že pro Jump server bude vhodnější implementace jako **kontejner**. Kontejner vyhovuje k nižší režii a spotřebě výpočetních zdrojů. Jeho implementace a replikace je rychlejší a efektivnější vzhledem k jeho funkcí. Avšak jeho nevýhodou jsou i některé omezené funkcionality oproti VMs, které by mohli mít vliv při implementaci. V případě nasazení Jump serveru jako VM, by efektivita nemusela být až tak fatální, nebo využití HW zdrojů, pokud by se zvolila správná technika implementace.

# 2 Manuální implementace

V první části praktické části je prováděna manuální implementace do zvoleného scénáře autorem, a na základě výstupu testování je vybrána technologie pro Jump server. V této podkapitole je popsáno manuální vytváření experimentálního prostředí a zprovoznění Jump serveru do vytvořené topologie. Nejprve je Jump server testován jako kontejner a následně jako virtuální stroj. Také je popsána OpenStacková funkce, která umožňuje přístup k VMs přes plovoucí IP adresy.

## 2.1 Topologie scénáře

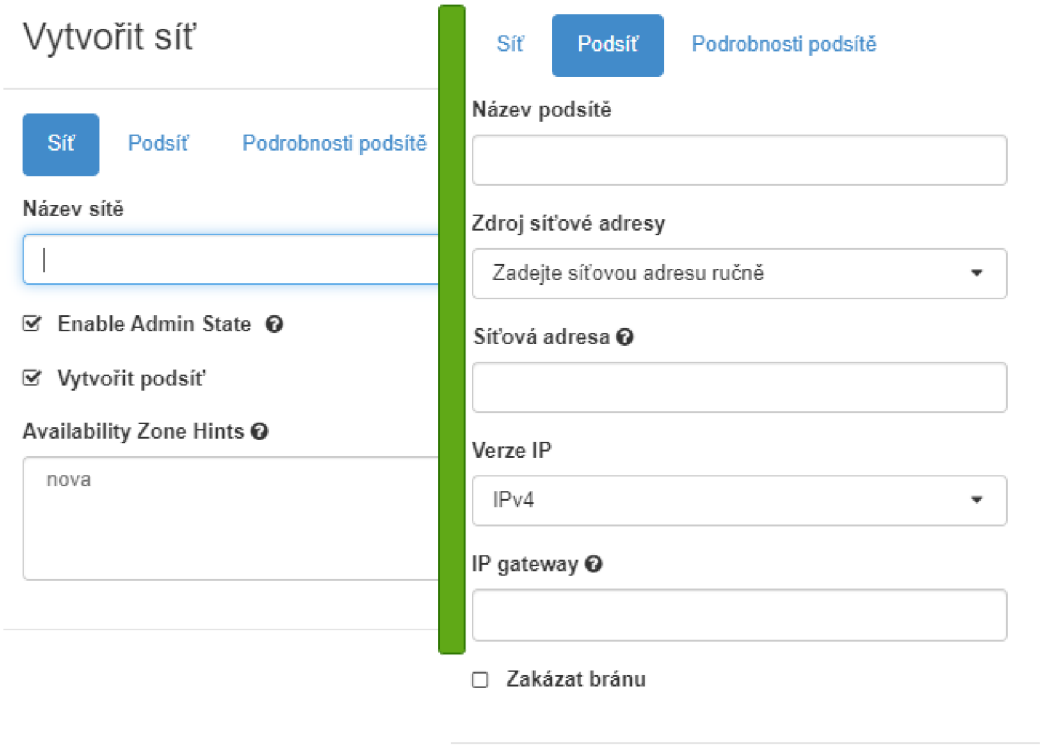
Topologie experimentálního prostředí, ve kterém bude testován Jump server se skládá ze dvou virtuálních strojů, dvou sítí a dvou směrovačů 2.1. Jump server bude poskytovat připojení k virtuálním počítačům za NAT sítě směrovačů pro uživatele z fyzické sítě, kteří jsou připojeni do virtuální Kybernetické arény přes VPN.




Obr. 2.1: Topologie experimentálního prostředí

Při vytváření topologie bylo nutné nejprve vytvořit sítě Network-1 a Network-2. Sítě jsou vytvářeny v prostředí ovládacího panelu OpenStacku, kde v hlavní záložce Projekt je nutné otevřít záložku Síť a v ní Sítě. Po otevření záložky Sítě je možné vybrat v pravém horním rohu volbu vytvořit síť. Po otevření je zobrazen formulář viz obrázek 2.2.

Do formuláře se zadává název sítě, a zda bude vytvořena podsíť. Při vytváření podsítě (vpravo na obrázku 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. Přes poslední záložku Podrobnosti podsítě je možné konfigurovat služby DHCP.




Obr. 2.2: Formulář k vytváření sítí

### 2.1.1 Tvorba směrovačů

K vytvoření dvou topologií je potřeba vytvořit směrovače. Směrovač se vytváří ve stejné záložce jako sítě. Pro vytvoření směrovače stačí zadat název a přidělit jej do externí sítě. V tomto případě jsou směrovače připojeny do sítě public. Po vytvoření je nutné přidat v rozhraní směrovačů vytvořené podsítě sítí, do kterých budou připojeny VMs.

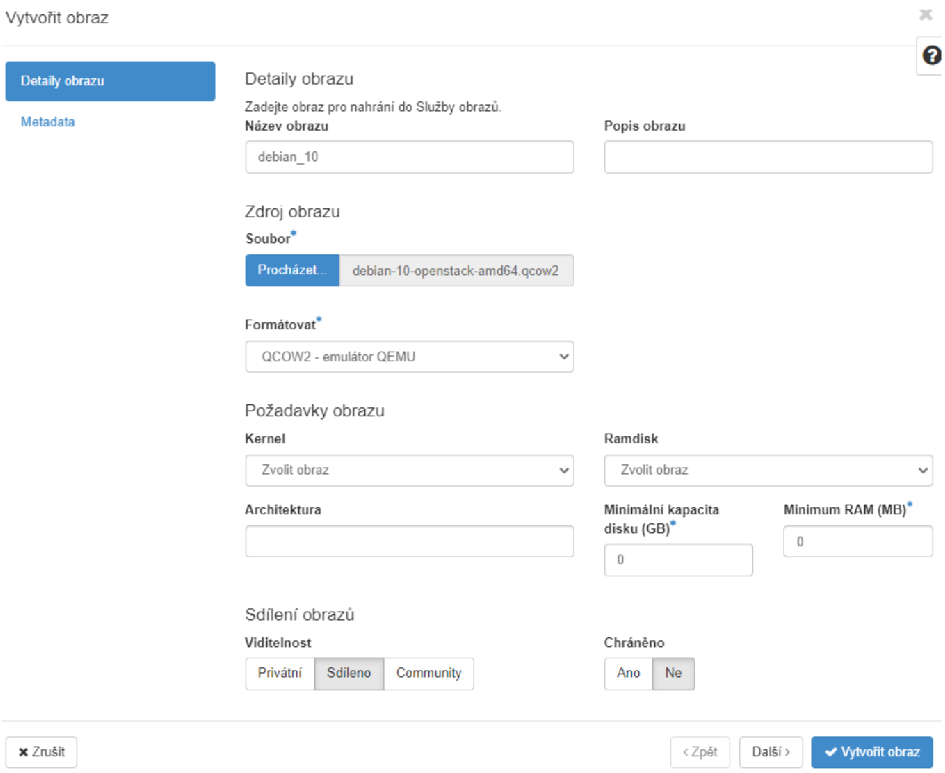
### 2.1.2 Tvorba Instance

Tvorba instancí 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 předsdílený na úložišti. Typ flavoru byl vybrán s přiřazením jednoho virtuálního procesoru, 1 GB RAM paměti a 10 GB místa na disku.

### 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 cloudovou službu OpenStack, pro importování do úložiště. Formulář pro přidávání obrazů je zobrazen na obrázku 2.3.




Obr. 2.3: Formulář k přidání obrazu do OpenStacku

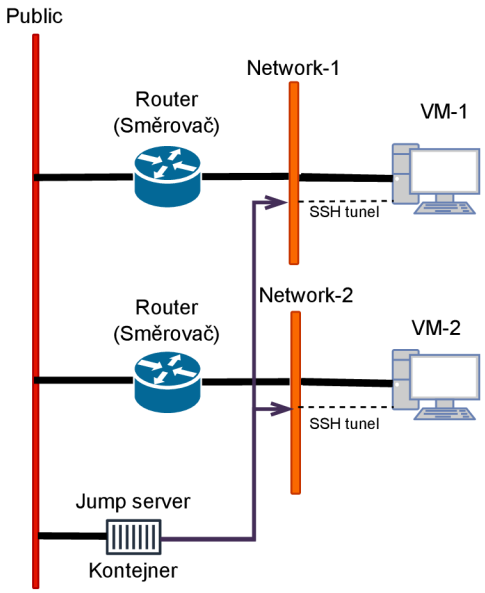
Pro úspěšné vložení je nutné vyplnit název obrazu, zdroj a podle typu vybrat formátování. V konkretním případě byl zvolen emulátor QEMU na qcow2 formát disku. Další nutnou položkou je vyplnění minimální kapacity disku a paměti RAM, avšak ve výsledku přidělených HW zdrojů rozhoduje typ instance při použití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

## 2.2 Analýza způsobů k tunelování

Jump server je možné implementovat dvěma způsoby. Jedním z nich je za využití kontejnerizace a vytvořit jej za pomocí kontejneru. Druhý způsob je implementace Jump serveru jako virtuálního stroje. Jelikož se jedná o dost odlišné technologie a každá má své výhody i nevýhody. Možnosti implementace jsou popsány v následujících dvou podkapitolách dle technologie.

### 2.2.1 Připojování k jednotlivým virtuálním sítím

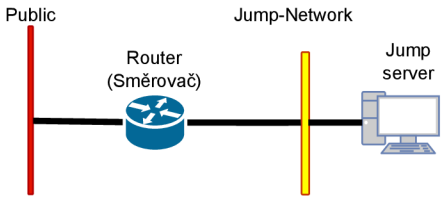
Pro síťování v OpenStacku je používána služba Neutron, ke které je i připojen ovladač Kuryr. Tento ovladač spravuje síťové nastavení kontejnerů. Hlavní výhodou tohoto propojení je možnost kontejnery připojovat do vytvořených sítí Neutronem. Tato kompatibilita bude využita pro manuální implementaci Jump serveru, kde kontejner bude připojen do veřejné sítě public a od DNS serveru obdrží IP adresu. Dále kontejner bude připojen k virtuálním sítím v kterých se nachází VMs, ke kterým má být vytvořen SSH tunel. Grafickou implementaci Jump kontejneru a jeho funkcionalitu je možné vidět na obrázku 2.4. Přes kontejner bude možné připojení do VMs. Nevýhodou této implementace bude nutné ošetření duplicitních IP adres, jelikož Jump server by byl připojován do několika scénářů, ve kterých jsou stejné sítě se stejnými IP adresy.




Obr. 2.4: Topologie Jump serveru jako kontejner.

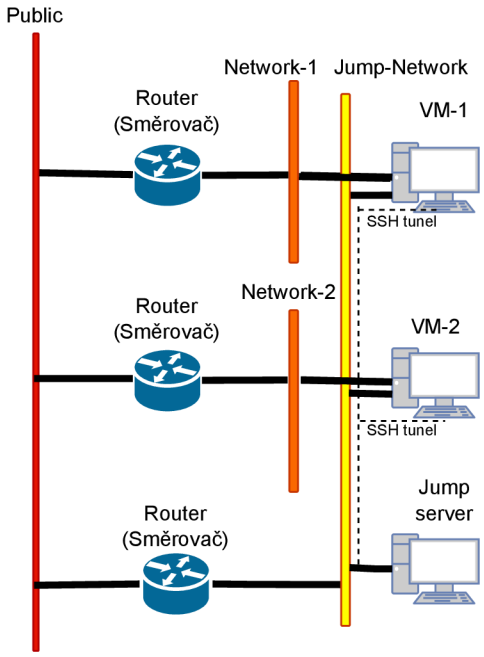
### 2.2.2 Připojování přes sdílenou virtuální síť

Hlavním rozdílem pro druhý způsob připojování bude vytvořená topologie, která se bude skládat ze směrovače, Jump sítě a virtuálního stroje Jump serveru. Celá topologie je vyobrazena na obrázku 2.5. Tento způsob bude fungovat na principu opačném a bude využívána i síť, ve které je Jump server. V tomto případě je způsob vyobrazen pro Jump server jako virtuální stroj.




Obr. 2.5: Topologie Jump serveru.

Druhý způsob funguje na připojení VM do Jump sítě, ve které je připojený Jump server. Po připojení VM do sítě je možné vytvořit, na základě nově přidělené IP adresy, SSH tunel přes Jump server. Tento způsob je graficky znázorněný na obrázku 2.6. Hlavní výhodou tohoto způsobu je ošetření problému duplicity adres, který vzniká v první metodě. Taktéž se i v rámci automatizace nabízí lepší řešení při implementaci, kde bude na základě podmínky VM připojena do sítě s Jump serverem. Vzhledem k těmto výhodám je tento způsob vybrán pro manuální implementaci a následně i pro automatizaci. Na obrázku 2.5 je znázorněna impelemntace Jump serveru jako VM, avšak je možné stejným způsobem implementovat Jump server jako kontejner. To platí i pro první způsob, kde je na obrázku 2.4 topologie s kontejnerem, tak je možné použít Jump server jako virtuální stroj, který je připojení přímo do public sítě..




Obr. 2.6: Druhý způsob implementace Jump serveru.

### 2.2.3 Přesměrovávání komunikace pomocí OpenStacku

Další možností pro SSH tunelování ke vzdáleným instancím je využít funkce OpenStacku. Tato funkcionalita je založena na přidělování plovoucích IP adres. Jedná se také o takzvané veřejné IP adresy sloužící pro vzdálený přístup z fyzické sítě do cloudového prostředí k instancím. Samotný Jump server bude mít právě přidělenou veřejnou IP adresu pro přístup z fyzické sítě, a tím bude umožněno skrze něj tunelovat připojení k virtuálním serverům.

Pro přidružení plovoucí IP adresy je nutné jít do sekce Instance v grafickém prostředí OpenStacku. U každé vytvořené instance je možné provést nějakou akci. Po otevření seznamu akcí stačí kliknout na akci s názvem Přidružit plovoucí IP. Zde vybereme plovoucí IP adresu, kterou chceme přidružit a potvrdíme. Ještě před přidružením adres je nutné v záložce Sít v sekci Plovoucí IP adresy požádat o přidružení plovoucí IP adresy z externí sítě OpenStacku. Obdržené veřejné adresy z externí sítě je možné asociovat k instancím, pod podmínkou, že se instance nachází v NAT síti za jedním směrovačem. Pro virtuální stroje v NAT sítích s rozsáhlou topologií o dvou a více směrovačích platí, že není možné instancím přidružit plovoucí IP adresu.

Jelikož s implementací Jump serveru jako virtuálního stroje budou VMs přepojovány do Jump sítě, která je za jedním směrovačem, bude možné instancím přidružovat veřejné adresy. Nevýhodou veřejných adres je jejich omezená kapacita. V rámci

automatizace bude tato funkcionalita taktéž zprovozněna a ověřena.

## 2.3 Techniky tunelování SSH

SSH tunelování na Jump serveru k VMs je možné implementovat několika způsoby. Celkem máme čtyři techniky k tunelování spojení přes Jump server.

- **Přesměrování lokálního portu (Local port forwarding)** – 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. Jedná se přesměrování komunikace z našeho lokálního počítače přes určitý port Jump serveru na cílový server ve vzdálené síti.
- **Přesměrování vzdáleného portu (Remote port Forwarding)** – Tato technika umožňuje vytvořit SSH tunel mezi Jump serverem a cílovým serverem ve vnitřní síti. Komunikace je opět šifrována. Je možné přesměrovat komunikaci z Jump serveru na lokální počítač pomocí specifického portu.
- **Dynamické přesměrování portu (Dynamic port Forwarding)** – Jedná se o SSH tunelování přes takzvaný SOCKS proxy server na lokálním počítači, který využívá SSH tunel a Jump server. Je umožněno směrování různé komunikace přes proxy server do vzdálené sítě prostřednictvím Jump serveru.
- **Přesměrování agenta (Agent Forwarding)** – Jedná se spíše o SSH agenta pro ukládání SSH klíčů a poskytování autentizace pro běh SSH relace. Po prvním SSH spojením si agent ukládá klíče popřípadě hesla, která byla použita pro SSH tunel. SSH autentizační agent se po aktivaci přesměrování stává dostupným pro vzdálený server, na kterém se připojujeme, a tím se stává sdíleným i pro vzdálený server. Lze použít autentizačního agenta na vzdáleném serveru pro autentizaci na dalších serverech, ke kterým máme přístup. Hlavní podmínkou je akceptování autentizace přes SSH klíče.

### 2.3.1 Přesměrování lokálního portu

Způsob techniky pro SSH tunelování bude pomocí lokálního přesměrování portu na Jump serveru. 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, k instanci, ke které má být připojen. Nejtypičtější příklad pro lokální přesměrování:

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

Po zadání se server připojí k 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.

## 2.4 Jump server jako kontejner

Před implementací Jump serveru jako kontejneru bylo nutné provést analýzu obrazů, který budou použit pro kontejnerizovaný server. Následně vybraný obraz je použitý pro vytváření vlastního kontejneru.

### 2.4.1 Analýza obrazů

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

Z tabulky je možné vyčíst, že obrazy s nejnižší velikostí se pohybují okolo 15 MB. Tyto modifikované obrazy využívají základní obraz Alpine Linux. Používaný obraz Alpine patří pod linuxovou distribuci a byl vytvořen pro efektivní využití HW zdrojů. Jeho základní velikost je okolo 5 MB. Vzhledem k jeho vlastnostem má vhodné využití jako základní obraz pro Jump server.

### 2.4.2 Tvorba vlastního kontejneru pro Jump server

Tvorba vlastního modifikovaného obrazu pro kontejner Jump serveru byla prováděna ve virtuálním stroji 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 standardně 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:

```
\newpage
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
```

### 2.4.3 Dodatečná úprava topologie pro Jump server

Jelikož byla vybrána varianta dle kapitoly 2.2.2, 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 2.2.

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

Tab. 2.2: Informace o Jump síti

Také byl vytvořen směrovač, jenž byl připojen do sítě public. Přes rozhraní směrovače byla přidána podsíť Jump sítě.

### 2.4.4 Konfigurace kontejneru

Při tvorbě kontejneru byly nastavovány konfigurace, jež jsou vepsány v tabulce 2.3. Sítě byly nastavovány podle potřeb k danému scénáři. Další možná nastavení zůstala ve výchozím stavu, jelikož v rámci testování funkčnosti nemají žádný vliv.

| 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. 2.3: Konfigurace kontejneru při vytváření

## 2.5 Zprovoznění Jump kontejneru

V následujících podkapitolách je popsán postup testování funkčnosti Jump serveru jako kontejneru. 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.

### 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 2.4 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. 2.4: IP adresy jednotlivých VMs

Ověření funkčnosti v rámci zvolené topologie, která je tvořena dvěma sítěmi, byl vytvořený kontejner 2.1 přiřazen do sítě Jump-Network, Network-1 a Network-2. Kontejneru je následně přiřazena plovoucí IP adresa z fyzické sítě public, a to 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@192.168.90.198
```

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
```

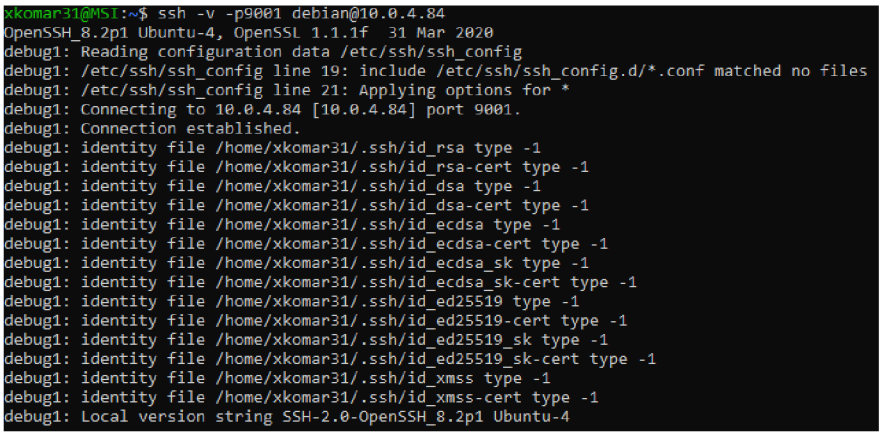
V obou případech se nepodařilo připojit k daným VM. Aby byla zjištěna příčina, byl v obou případech příkaz k připojení rozšířen o možnost -v, jež umožňuje vypsat průběh 2.7 SSH připojení. Připojení k portu vždy proběhlo správně, ale klient neobdržel zprávu v prostém textu ohledně verze protokolu od VMs.

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

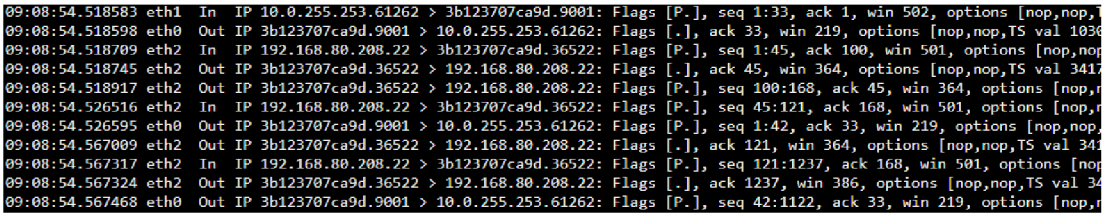
Tento incident 2.7 nastával v obou případech. Další kroky vedly k hledání chyb. Nejprve proběhl test, zda probíhá SSH propojení mezi VM a serverem. K tomuto testu byl využit příkaz tcpdump, jenž slouží pro analýzu komunikace v síti. Nejprve bylo nutné manuálně stáhnout službu do Jump serveru a poté byla 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 2.8.

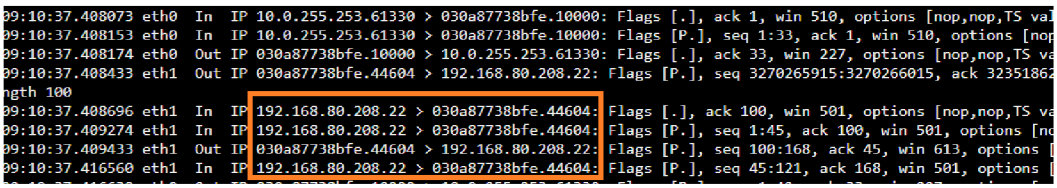



Obr. 2.7: Neúspěšné připojení




Obr. 2.8: Odposlech neúspěšného připojení

V případě úspěšného připojení, jež proběhlo v rámci jedné sítě musí vypadat spojení jako na obrázku 2.9, kde je možné vidět komunikaci od VM, která přes Jump server předala verzi SSH protokolu.




Obr. 2.9: Odposlech úspěšného připojení

### 2.5.1 Vyhodnocení funkčnosti

Připojení k VMs přes Jump server bylo neúspěšné. Hlavní příčina tohoto problému byla nalezena během porovnávání směrovacích tabulek, které jsou na Jump serveru nastavené OpenStackem. V neúspěšném případě byla nastavena výchozí brána v podobě několika záznamů, které pokrývají všechny směrovače, které jsou připojeny

k Jump síti. Aby bylo připojení přes Jump server úspěšné je nutné nastavit výchozí bránu tak, aby směrovala na směrovač připojený k fyzické síti, a nikoliv na některý ze směrovačů ve virtuálních sítích.

Přenastavit výchozí bránu v kontejneru Jump serveru je samotným OpenStackem zakázané, a jelikož je samotná Kybernetická aréna v neustálém vývoji, povolit či přepracovat některé funkce v OpenStacku by mohlo mít špatný vliv na samotný chod Arény, popřípadě na některé její funkce. Taktéž i v rámci zabezpečení kontejnerů jsou některé funkce zakázané. Vzhledem k zmíněným důvodům nemohl být Jump server kontejnerizován, proto v kapitole 2.6 je popsána implementace Jump serveru v podobě VM.

## 2.6 Implementace Jump serveru jako VM

V následujících podkapitolách je popsán postup k manuální implementaci Jump serveru jako VM. Na základě analýzy obrazů byl vybrán ten, jenž splňoval podmínky pro VM Jump server.

### 2.6.1 Obraz pro Jump server

K efektivnímu nakládání výpočetních zdrojů byla provedena analýza diskových obrazů pro Jump server. Analýza obrazů proběhla mezi nejznámějšími linuxovými distributory jako jsou CentOS, Ubuntu, Debian a Fedora. Pro instanci Jump serveru byl vybrán obraz od distributora Ubuntu, který se specializuje i na minimální velikosti disků ve formátu qcow2.

### 2.6.2 Vytvoření Jump serveru

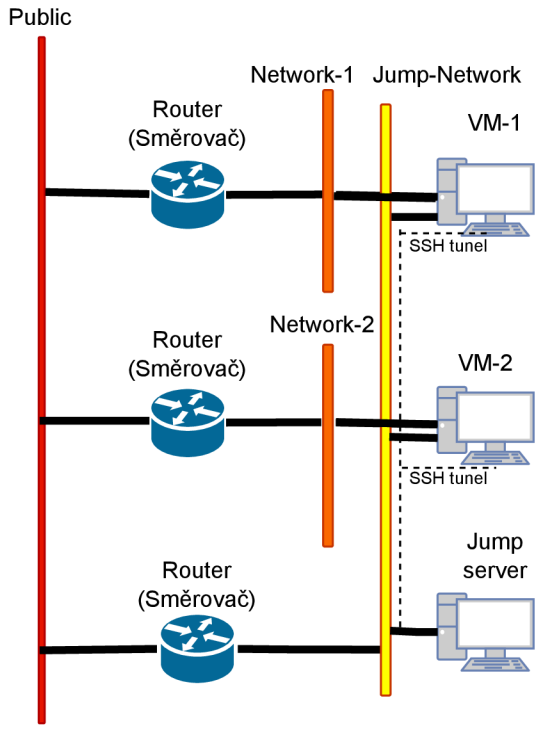
Další krok k implementaci Jump serveru jako VM vedl k vytvoření instance, která byla připojena do sítě Jump-Network. Aby byl Jump server dostupný z fyzické sítě, byla mu přidělena plovoucí adresa, a také je využívána k vytváření tunelů k instancím. Podrobnosti o Jump serveru jsou napsány v tabulce 2.5.

| Název instance | Jump-server |
|---|---|
| Obraz instance | jump-image |
| Flavor | arena.1-1.5 |
| Síť | Jump-Network |

Tab. 2.5: Informace o Jump serveru

### 2.6.3 Příprava scénáře

Topologie scénáře zůstává obdobná jako 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 připojovány do již vytvořené Jump sítě. Celá topologie i s Jump serverem je vyobrazena na obrázku 2.10.




Obr. 2.10: Topologie s umístěným Jump serverem.

Připojené VMs do sítě obdržely od DHCP služby IP adresy. Je možné 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.

| | |
|---|---|
| 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

### 2.6.4 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 serveru z jiného zařízení, 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
```

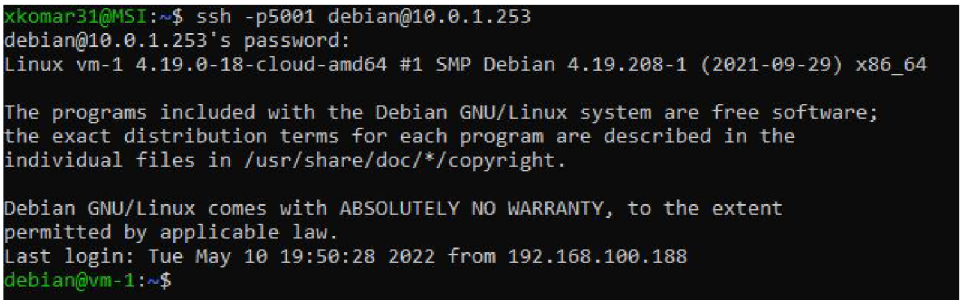
Pro zadání každého příkazu je nutné zadat heslo k jednotlivým VMs. Na zvolených portech vznikne SSH tunel k VMs přes Jump server. K připojení k VM-1 je nutné na straně klienta zadat:

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

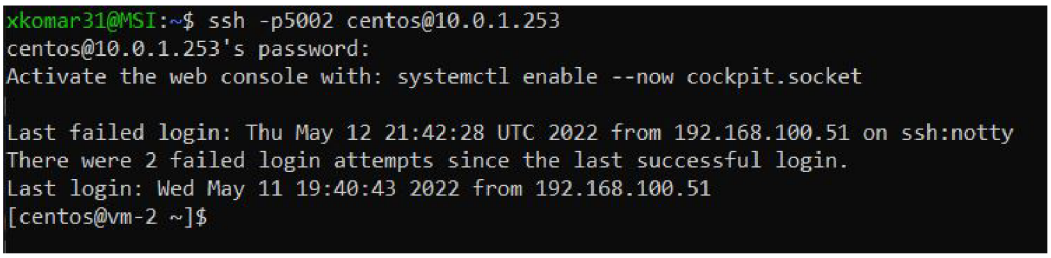
V případě připojení k VM-2:

```
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 2.11 a v případě VM-2 2.12.




Obr. 2.11: Připojení přes Jump server k VM-1.




Obr. 2.12: Připojení přes Jump server k VM-2.

### Vyhodnocení funkčnosti

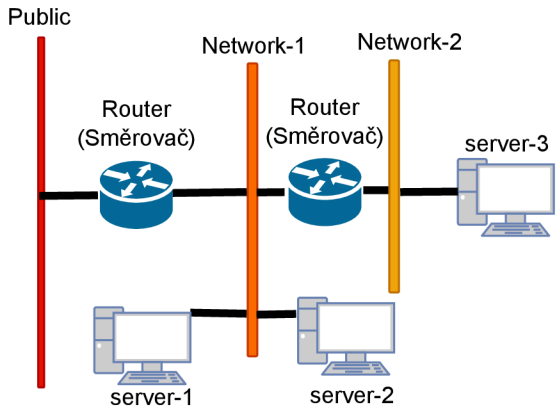
Při použití Jump serveru jako VM připojení k virtuálním strojům v topologii bylo úspěšně tunelováno přes implementovaný Jump server. Taktéž VMs nejsou nijak omezovány v modifikaci a používání nástrojů oproti kontejnerů, které některé modifikace nepodporují. Jelikož Jump server jako VM funguje bez komplikací a umožňuje širší spektrum modifikací, bude využit pro automatizaci v Kybernetické aréně.

# 3 Automatizace

Pro automatizaci je použit veřejně dostupný software Ansible, který nabízí využití v automatizaci procesů na cloudových platformách. V následujících podkapitolách je popsán postup automatizace Jump serveru.

## 3.1 Příprava prostředí

Topologie pro scénáře v Kybernetické aréně je generována pomocí šablony, kterou autor práce obdržel od vedoucího práce zabývající se problematikou automatizací scénářů ve virtuální aréně. Šablona generuje scénář na základě proměnných v konfiguračním souboru, jenž definuji strukturu topologie pro daný scénář. Pro automatizaci je struktura topologie generována s třemi virtuálními stroji. Celou topologii je možné vidět na obrázku 3.1. Hlavním cílem bude zprovoznění SSH tunelu k serveru3.




Obr. 3.1: Topologie scénáře pro automatizaci.

### 3.1.1 Databáze

Pro správu dat a informací s kterými aplikace pracuje musela být vytvořena databáze. Konkrétně se jedná o MySQL databázový systém, který slouží pro správu databází. Hlavní výhodou je rychlost a výkon, kde systém dokáže rychle a efektivně zpracovávat data. Jedná se o open-source software, tudíž je bezplatný a správce si ho může upravovat podle potřeb.

V případě nutnosti nabízí i zabezpečení jako je tvorba uživatele s heslem a popřípadě šifrování dat. Je kompatibilní s různými operačními systémy včetně Linuxu, Windows a macOS. V tomto případě je instalován v linuxovém systému CentOS, na kterém běží Ansible server. Existuje i webová stránka s podporou a dokumentací pro MySQL databázi, kde jsou potřebné informace jak databázi nainstalovat a jak se s ní pracuje.

Autor práce tvorbu databáze automatizoval. Automatizace je nakonfigurována na již zmíněný linuxový systém CentOS. Automatizace instaluje MySQL databázi a zároveň v ní vytváří potřebné části, které jsou využívány při automatizaci Jump serveru se scénáři.

Databáze je implementována pomocí role `database`, která obsahuje playbook main.yml, jenž obsahuje potřebné úkoly k automatizaci. Role `database` je i dál používána v rámci automatizace scénářů, jelikož data o scénářích jsou ukládána a v případě smazání scénářů jsou i smazána z databáze. Role se volá přes playbook `bootstrap-db.yml`. Před inicializací automatizace je nutné mít nakonfigurovány proměnné v konfiguračním souboru `all.yml` v adresáři `group_vars`. Databáze je vytvářena na Ansible serveru s konfiguračními údaji, jenž je možné vidět:

```
db_ssh_user: 'root' # Název uživatele v Ansible serveru
db_ssh_ip: 'localhost' # Ansible server je localhost
db_name: Areny # Název databáze
table_name: scenare # Název tabulka v databázi
db_port: '3306' # Port
db_user: 'dbadmin' # Uživatel databáze
db_pass: 'admin' # Heslo uživatele
remote_addr: 0.0.0.0 # Nastavení adres pro přístup k DB
bootstrap: true # Potřebné zahájit před automatizací
```

Pro inicializaci je použit příkaz:

```
ansible-playbook bootstrap-db.yml
```

V následujícím seznamu jsou popsány úkoly, které role `database` postupně vykonává:

1. **Instalace balíčků Pythonu** – Je provedena instalace balíčku Python 3 pip, což je nástroj pro správu balíčků v jazyce Python.
2. **Instalace pip modulu PyMySQL** – Pro umožnění komunikace mezi Pythonem a MySQL databází je nutná instalace modulu PyMySQL.

3. **Ověření statusu služby MySQL** – Pomocí modulu Ansible je zkontrolováno, zda je služba MySQL spuštěna, a pokud není, je spuštěna.
4. **Konfigurace vzdáleného přístupu** – Je provedena konfigurace vzdáleného přístupu k databázi. Konfigurační soubor mysqld.cnf.j2 je upraven podle šablony a uložen na cílovém serveru.
5. **Restart služby** – Služba MySQL je restartována, aby došlo k provedení změn v konfiguraci serveru.
6. **Vytvoření databáze** – Je vytvořena databáze s názvem z konfiguračního souboru. Modul `community.mysql.mysql_db` je využit pro správu databáze a je zkontrolováno, zda databáze již existuje.
7. **Vytvoření uživatele** – Je vytvořen uživatel databáze s přístupovým heslem a přidělena mu jsou příslušná oprávnění pro práci s databází.
8. **Vytvoření tabulky** – Je vytvořena tabulka za pomocí komunitního modulu `community.mysql.mysql_query`, který provádí SQL dotaz. Tabulka obsahuje specifické názvy sloupců pro ukládání dat při tvorbě scénářů s Jump serverem.

### 3.1.2 Analýza implementace Jump serveru

Jump server bude automatizován na základě své topologie vytvořené při manuálním zprovoznění. Tato topologie bude automatizována do generovaných scénářů a na základě podmínky, jenž udává nutnost připojení k vzdálenému virtuálnímu stroji v Kybernetické aréně bude vytvořen SSH tunel pomocí Jump serveru.

## 3.2 Automatizace procesů

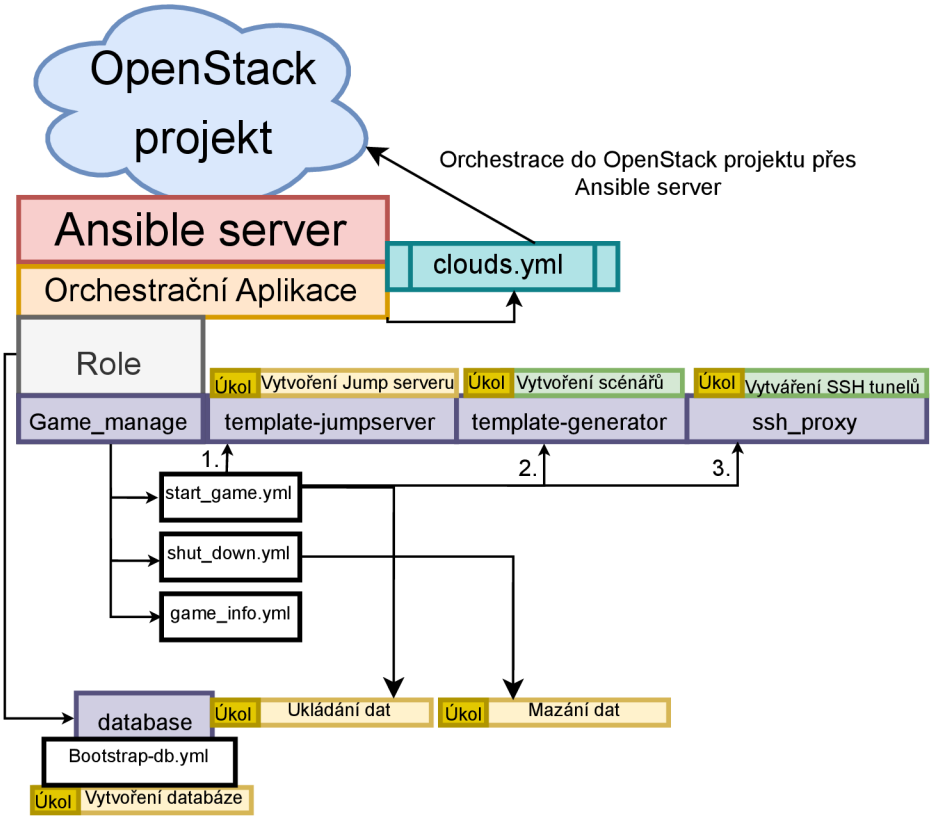
Automatizace procesů nutných k zprovoznění a automatizaci Jump serveru a scénářů je popsána v následujících podkapitolách. Pro automatizaci je používán nástroj Ansible, který je volně dostupný veřejnosti. Podrobnější popis nástroje Ansible je v následující podkapitole. A také v dalších podkapitolách této kapitoly je popsán postup k automatizaci Jump serveru a scenářů.

### 3.2.1 Ansible

Jak již bylo zmíněno Ansible je volně dostupný softwarový nástroj v podobě open-source software. V používaném prostředí je Ansible nasazen jako virtuální server, který běží v cloudovém projektu OpenStacku. Jedná se o virtuální stroj s operačním systémem Linux s doinstalovanými balíčky v podobě samotného Ansible nástroje a jiných balíčků, jenž slouží k automatizaci procesů na OpenStacku. Virtuální Ansible server už byl nakonfigurován v obdrženém OpenStack projektu, v kterém probíhá automatizace. Na obdrženém serveru byla taktéž už vytvořená složka s potřebnými soubory k takzvané orchestraci scénářů.

Jedná se o Heat šablonu, která podle zdrojového kódu definuje infrastrukturu automatizovaného scénáře. Součástí Heat šablony je hlavní playbook, který odkazuje na další playbooky napsané v jazyce YAML. Tyto playbooky obsahují úkoly s instrukcemi automatizace scénáře za použití Heat šablony. Nemusí jít jen o úkol pro použití Heat šablony, ale také o úkol pro instalování, ukládání, vytváření, záleží k čemu daný úkol má sloužit. V anglické terminologii je tento úkol či úloha označována jako task v množném čísle tasks.

Autor práce provedl pouze aktualizaci Ansible serveru a některých funkcí. Obdržené soubory byli taktéž upravovány a rozšířeny v rámci zprovoznění automatizace Jump serveru na základě generované Heat šablony a v případě aktualizace bylo možné používat novější funkce. Celá architektura implementace Ansible serveru a funkčnosti je graficky znázorněna na obrázku 3.2.

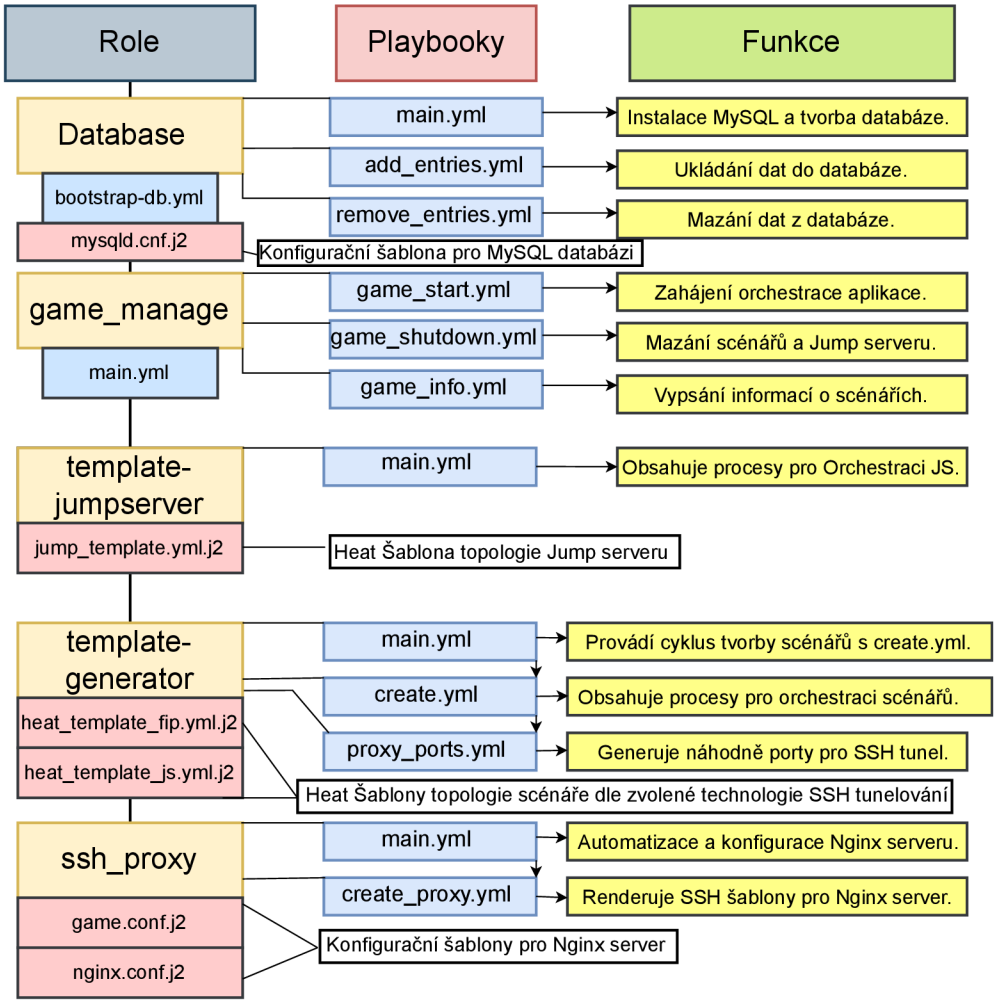



Obr. 3.2: Schéma provázanosti Ansible rolí v orchestrační aplikaci.

Nejpodstatnějším souborem je `clouds.yaml`, který obsahuje údaje pro autentizaci k připojení k OpenStack projektu, aby byl schopný Ansible komunikovat a pracovat s cloudovým prostředím. Tento soubor je i možné přímo stáhnout z prostředí OpenStack v sekci API přístup. Do souboru je nutné doplnit heslo k uživateli OpenStack projektu k zajištění přístupu Ansible serveru.

### 3.2.2 Architektura procesů

Automatizační struktura je tvořena rolemi, které obsahují playbooky. Jak již bylo zmíněno tyto playbooky obsahují instrukce úkolů v jazyce yaml podle kterých se vykonávají procesy pro automatizaci na platformě OpenStack. Taktéž některé playbooky v jednotlivých rolích můžou odkazovat na další role, ve kterých jsou další playbooky s úkoly. Jak vypadá taková architektura rolí a playbooků v případě automatizace Jump serveru a scénářů je zobrazeno na obrázku 3.3.




Obr. 3.3: Architektura implementace procesů.

Na obrázku můžeme vidět celou strukturu automatizační aplikace a kolik bylo potřeba vytvořit rolí s jednotlivými playbooky a procesy, bylo vytvořeno celkem pět rolí. Hlavní role `game_manage` obsahuje playbooky, přes které se orchestrační aplikace ovládá, úkoly v playbooku například odkazují k dalším rolím k vytvoření scénářů s Jump serverem, nebo obsahují procesy sloužící k výpisu důležitých informací o každém vytvořeném scénáři, popřípadě poslední třetí playbook k mazání

všech scénářů včetně Jump serveru.

Další čtyři role jako `database`, `ssh_proxy`, `template-generator` a `template-jumpserver` budou popsány v následujících podkapitolách níže, jelikož obsahují specifičtější playbooky s automatizačními procesy. Je nutný podrobnější popis pro konkrétní automatizovaný prvek danou rolí.

### 3.2.3 Automatizace Jump serveru

Funkci pro automatizaci Jump serveru má na starost role `template-jumpserver`. Podstatnou částí role je také Heat šablona, v které je napsána struktura topologie Jump serveru v jazyce yaml. Tuto šablonu volá jeden z procesů v playbooku pro automatizovanou implementaci do prostředí OpenStacku. Jakým způsobem topologie vypadá bylo vyobrazeno na obrázku 2.5 při manuální implementaci.

Ještě předtím, než je příkazy volána Heat šablona je generováno heslo pro Jump server. S velkou pravděpodobností je tato funkce navíc, ale je umožněn přistup na vzdálený virtuální Jump server pro administrátora v případě nutnosti a uživatelé scénářů nemají k heslu přístup. V rámci praxe by bylo možné zkusit použít útok hrubou silou pro nalezení hesla, heslo je totiž možné nastavit na jakoukoliv délku znaků. Hlavní playbook pro tvorbu Jump serveru je popsán v číselném seznamu:

1. **Zisk seznamu zásobníků** – Proces získá stávající seznam zásobníků a následně jej uloží do proměnné pro pozdější použití. Zdali v seznamu je již vytvoření zásobník Jump serveru, budou následující procesy přeskočeny.
2. **Generování náhodného hesla** – Pro Jump server je generováno heslo k přístupu.
3. **Vytváření z hesla Hash** – Dochází k tvorbě hashe z uživatelského hesla Jump serveru. Je zde využívána knihovna crypt pro jazyk Python.
4. **Generování šablony** – Dochází ke generování Heat šablony pro topologii Jump serveru. Šablona je generována za pomocí generačního souboru typu jinja2.
5. **Validace šablony** – Dochází k validaci šablony, neboli k ověření, zda má šablona správnou syntaxi a je platná k použití.
6. **Vytvoření zásobníku** – Dochází k tvorbě zásobníku z vygenerované šablony. Dochází k orchestraci Jump serveru s Jump sítí.
7. **Čekání na vytvoření** – Aplikace čeká, než se celá topologie Jump serveru vytvoří, aby mohla pokračovat v dalších úkolech.

### 3.2.4 Automatizace scénáře

Role pro automatizaci scénáře byla již vytvořena na Ansible serveru. Role byla pojmenována jako template-generator. Nejdůležitější částí role je Heat šablona, ve které je obsažen kód pro generování scénářů. Součástí šablony je konfigurační soubor celé orchestrace config.yaml, který obsahuje proměnné a na základě proměnných definuje topologii scénáře. V roli také již byly vytvořeny playbooky k automatizované implementaci scénářů. Celkový počet scénářů je možné definovat právě v souboru config.yaml.

Vzhledem k tomu, jakým způsobem role fungovala, autor práce musel některé funkce úkolů v playbooku změnit, doplnit a rozšířit. Heat šablona taktéž byla pozměněna z důvodu implementace Jump serveru do vygenerovaných scénářů. Jelikož autor práce využívá pro práci s daty databázi, bylo nutné toto rozšíření implementovat také do playbooků s procesy automatizace scénářů.

Soubor config.yaml také obsahuje podmínku, podle které má být vytvořen přístup k vzdálenému virtuálnímu stroji přes Jump server. Jedná se o podmínku ssh connection : yes. Na základě podmínky musela být upravena šablona pro připojování VM do Jump sítě, aby bylo možné vytvářet SSH tunel k VM.

Z důvodu rozšíření funkčnosti vytváření připojení k vzdáleným VM v OpenStacku byla vytvořena druhá obdobná šablona, s tím že po připojení není vytvořen SSH tunel přes Jump server, ale VM je přidělena veřejně plovoucí IP adresa. Jedná se o možnost za využití OpenStack funkcí, která je podrobně popsána v podkapitole 3.2.6. Pro lepší představu je popsána implementace v číselném seznamu:

1. **Vypsání informací o cyklu** – Dochází k vypsání informací o prováděném cyklu. Pomocí modulu debug se vypíše zpráva ohledně scénářů a počtu studentů.
2. **Generování náhodného hesla** – Pro virtuální servery je generováno heslo k přístupům. A následně je heslo ukládáno do proměnné `vm_password`. Generování hesel se opakuje podle počtu virtuálních strojů.
3. **Vytváření hashových záznamů** – Dochází k tvorbě Hashů z uživatelských hesel Jump serveru, jenž jsou ukládány v proměnné `passwords` a opakovaně se provádí pro každé heslo.
4. **Filtrace hesel** – Provádí se filtrace a ukládání generovaných hesel do proměnné `vm_password`.
5. **Filtrace hashů z hesel** – Úkol provádí filtraci a ukládání hashů z hesel do proměnné `passwords`
6. **Přiřazení proxy portů** – Pokud je nastaven mód SSH jako jump_server a generují se porty pro servery k nimž má být vytvořen SSH tunel. Pro generování portů úkol odkazuje na playbook `proxy_ports_yml`, jenž jeho proces generuje

porty. Porty jsou ukládány do proměnné `existing_ports`. Také dochází ke kontrole, zda už generovaný port neexistuje v databázi, aby nedocházelo ke konfliktům.

7. **Generování šablony pro orchestraci** – Podle módu dochází ke výběru generačního souboru jinja2, pro generaci šablony scénáře přes kterou pak dochází k orchestraci do projektu OpenStack.
8. **Validace šablony** – Dochází k validaci šablony, neboli k ověření, zda má šablona správnou syntaxi a je platná k použití.
9. **Vytvoření zásobníku** – Dochází k tvorbě zásobníku z vygenerované šablony. Dochází k orchestraci topologie scénáře.
10. **Čekání na vytvoření** – Aplikace čeká, než se scénář vytvoří, po vytvoření zásobníku aplikace pokračuje v dalších procesech.
11. **Restartování proměnných** – Dochází k resetu proměnných, aby byly připraveny pro novou iteraci dalších cyklů.

### 3.2.5 Automatizace SSH tunelů

U manuálního zprovoznění Jump serveru a SSH tunelování, bylo využito principu lokálního přesměrování portu. K efektivnímu vytváření SSH tunelů při automatizaci bylo nutné toto řešení optimalizovat. Pro zachování funkcionality při vytváření SSH tunelů, kde pomocí plovoucí IP adresy Jump serveru a přiděleného portu je možné se připojit k vzdálené VM, bude použit nginx server.

Nginx je webový server který převážně pracuje s protokoly jako jsou HTTP, SMTP, POP3 a SSL. Vzhledem k funkčnostem Nginx serveru a jeho využití nabízí i možnost přesměrovávání, která je potřebná i v případě vytváření SSH tunelů. Nginx server v Jump serveru bude přesměrovávat připojení k VM za pomocí přiděleného náhodného portu a plovoucí adresy Jump serveru. Taktéž je i potřebná IP adresa VM přidělená Jump sítí.

K automatizaci vytváření SSH tunelů z Jump serveru k VM byla vytvořena role ssh_proxy. Role obsahuje dvě šablony, jedna pro konfiguraci nginx serveru a druhá pro vytváření konfigurací SSH tunelů. Jsou zde obsaženy dva playbooky s procesy. První hlavní playbook s úkolem volá druhý playbook, kde v prvním jsou volány procesy pro zjištění proměnných a na základě toho je volán druhý playbook pro vytváření tunelů z proměnných a šablony.

K zprovoznění nginx serveru slouží tzv handlers, který obsahuje playbook s procesem k restartování služby nginx. Na závěr jsou všechny důležité informace uloženy v databázi. Popis playbooku ve formátu číselného seznamu:

1. **Získání informací o zásobníku Jump serveru** – Procesy v bloku získávají veřejnou a privátní IP adresu o Jump serveru, pro nakonfigurování proxy Nginx serveru.
2. **Import úkolů** – Pomocí importovaných úkolů playbooku `create_proxy.yml` dochází k generování šablon pro konfiguraci nginx serveru. Pro každý scénář je zvlášť generována šablona přes `game.conf.j2`. Playbook `create_proxy.yml` skrz proces odkazující na `add_entries.yml` ukládá informace do databáze.
3. **Vytvoření konfigurace pro Nginx server** – Dochází k vytvoření šablony pro konfiguraci služby Nginx serveru a také dochází restartování služby k provedení změn.

### Nginx server

U manuálního zprovoznění Jump serveru bylo využito techniky lokálního přesměrování portu. V případě automatizace je využito webového Nginx serveru, který je nakonfigurován na obdobný způsob, jako při směrování lokálního portu.

Nginx server bude naslouchat na daném portu, například na portu 4350. K přidělenému portu bude určen cílový server naslouchající na portu 22, který je pro SSH službu. Jak takový konfigurační soubor vypadá je možné vidět:

```
upstream Arena1-server3 {
    server 192.168.100.179:22; # Cílový server
}
server {
    listen 4350; # Port na kterém Nginx server naslouchá
    proxy_pass Arena1-server3; # Port přesměrovává komunikaci na cílový server.
}
```

Tyto konfigurační soubory jsou generovány pro každý scénář a jsou ukládány v adresáři `/etc/nginx/conf.d` Nginx serveru. V tomto případě je možné vidět konfiguraci pro scénář Arena1. Vygenerované soubory pro scénáře jsou vyobrazeny na obrázku 3.4. Výpis pochází přímo z automatizovaného Jump serveru.

```
player@arena-jump-server:/etc/nginx/conf.d$ ls
Arena1-1.conf  Arena2-2.conf
```

Obr. 3.4: Adresář vytvořených konfiguračních šablon.

### 3.2.6 Tunelování s využitím OpenStack funkcí

Samotný OpenStack nabízí možnost připojení z fyzické sítě k virtuální instanci, a to za pomocí plovoucích adres. Tuto funkcionalitu lze taktéž využít, a to i v našem případě kde používáme implementaci topologie Jump serveru s Jump sítí. K připojeným VM do Jump sítě je možné následně přidělit plovoucí adresu, přes kterou se studenti budou připojovat k instancím ve scénáři. Vzhledem k tomu, že je možné tuto funkci použít, byla taktéž automatizována.

Pro automatizaci této funkce bylo nutné rozšířit automatizační aplikaci o další Heat šablonu pro generování scénářů. V případě podmínky ssh connection : yes je VM také připojena k Jump siti, avšak už není vytvářen SSH tunel přes Jump server, ale zde VM obdrží plovoucí IP adresu, která je využívána k připojení do instancí. Tato metoda není běžně aplikována v rámci širší implementace scénářů či topologií a větší organizace převážně využívají Jump server. Vše má své výhody i nevýhody.

Jump server nabízí výhody bezpečnosti, kdy jsme schopni aplikovat bezpečnostní opatření pro sledování a logování přístupů k instancím a také máme pouze jediný možný přístup přes Jump server. V tom může být i nevýhoda, pokud Jump server selže nebude možný žádný přístup k instancím. V některých případech je i omezená kapacita plovoucích IP adres, což dokáže vyřešit Jump server.

Výhodou plovoucích IP adres je umožnění přístupu k internetu instancím a také zajištění přímého připojení z fyzické sítě k virtuálnímu stroji v OpenStacku. Konfigurace přidělování plovoucích IP adres je o něco jednodušší než celá konfigurace Jump serveru. Co se týče bezpečnosti je zde nevýhoda přímého přístupu k instancím, tudíž pokud nejsou správně zabezpečeny jsou ohrožené a není možná stejná úroveň sledování a logování jako u Jump serveru. V případě omezeném množství plovoucích IP adres hrozí vyčerpání kapacit.

## 3.3 Funkce orchestrační aplikace

V následujících podkapitolách bude popsán postup manipulace orchestrační aplikací pro automatizovanou implementaci Jump serveru a scénářů. K tomu budou také podrobněji popsány jednotlivé playbooky s úkoly, které se vykonávají souvisle na sobě.

### 3.3.1 Konfigurace automatizace

V aplikaci je několik konfiguračních souborů, jedná se také o soubory typu .yml nebo .yaml. Jejich koncovka je stejná jak v případě playbooků, tudíž na první pohled se může zdát, že jde o playbook s automatizačními procesy, avšak vně jsou definovány proměnné v rámci orchestrační aplikace.

Celkem aplikace obsahuje tři konfigurační soubory. První z nich je běžně používaný, jelikož je součásti funkcí Ansible vytvořený v adresáři `group_vars`, a je využíván ke konfiguraci globálních proměnných závislosti na hosta. V tomto případě jde o tzv. implicitního hosta, kde se automaticky proměnné pro všechny hosty aplikují stejně a definuje se souborem all.yml. V playbooku je pak možné na tyto globální proměnné odkazovat.

Druhým konfiguračním souborem je `config.yml` v kterém je definována topologie scénáře s vysázenými proměnnými. Jedná se o uživatelský konfigurační soubor, který byl již vytvořený v adresáři aplikace. Posledním třetím souborem je `site.yaml`, ve kterém se odkazuje na uživatelský konfigurační soubor `config.yml` a také odkazuje na roli pro ovládání orchestrační aplikace. Také to optimalizuje příkazy pro provedení orchestrace, smazání zásobníku nebo výpis dat.

### 3.3.2 Spuštění aplikace

Pro spuštění automatizace byl vytvořen playbook `game_start.yml`, přes který se spouští orchestrační aplikace. Playbook obsahuje procesy odkazující na ostatní role s playbooky, v nichž jsou obsažené procesy pro orchestraci celé struktury Jump serveru se scénářem. Aplikace obsahuje dva módy:

- **Jump_server** – V případě módu jump_server, který je nutný nastavit v konfiguračním souboru, je tunelování k VMs vytvářeno přes Jump server.
- **Floating_ip** – V druhém případě je možné mód nastavit na floating_ip, kde se používají plovoucí adresy k přístupu na VMs.

Automatizace se spouští následujícím příkazem:

```
ansible-playbook site.yaml -e operation=game_start
```

Aplikace vykoná orchestraci za pomocí playbooku `game_start.yml`. Po zadaní příkazu se vykonávají postupně všechny úkoly z playbooků, tak jak byly popsány hierarchií v architektuře. Když už je orchestrace dokončena je možné si potřebné informace vypsat v příkazovém řádku. K vypsání informací je používán playbook `game_info.yml` s příkazem:

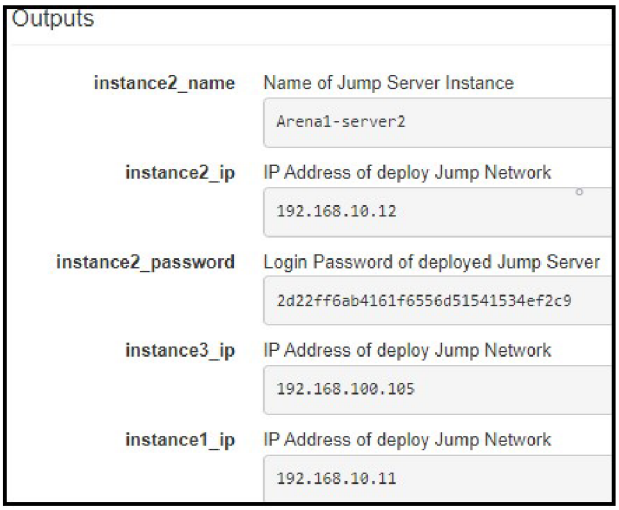
```
Ansible-playbook site.yaml -e operation=game_info
```

Informace ohledně orchestrace jsou také dostupné přímo ve webovém rozhraní OpenStacku. Administrátor může v záložce Orchestrace zobrazit všechny vzniklé

| Název | uživatel | IP adresa | Port | Heslo |
|---|---|---|---|---|
| Arena1-server3 | player | Není důležitá | 4350 | 0d0547d19c72 |
| Arena2-server-3 | player | Není důležitá | 8182 | 30f3bcb7b626 |
| Jump server | player | 10.0.4.76 | 22 | 36ef79a38e27 |

Tab. 3.1: Údaje potřebné k připojení do VMs.

zásobníky (stacky) scénářů a Jump serveru. V každém zásobníku je sekce Přehled se všemi informacemi, které jsou totožné s výpisem přes příkaz. Ukázku výpisu dat v přehledu je možné vidět na obrázku 3.5.




Obr. 3.5: Úkázka výpisu dat z přehledu OpenStacku.

### 3.3.3 Tunelování přes Jump server

Aplikace byla spuštěna s módem jump_server a byly vygenerovány dva scénáře včetně topologie Jump serveru s Jump sítí. V tabulce 3.1 jsou vepsány údaje o obdržené IP adrese pro Jump server a portech přiřazených k VMs. K připojení na vzdálenou instanci je důležitá přidělená plovoucí adresa Jump serveru a port, na kterém probíhá SSH tunelování. K ověření autentizace je využíváno heslo.

Zkouška, zda je možné se přes Jump server připojit na virtuální server scénáře, proběhla přes funkci WSL (Windows Subsystem for Linux). Jedná se o funkci v operačním systému Windows, která umožňuje spouštět distribuce Linuxu na systému Windows bez nutnosti instalování virtuálních strojů. Je možné tak spouštět linuxové příkazy či aplikace v příkazovém řádku Windows.

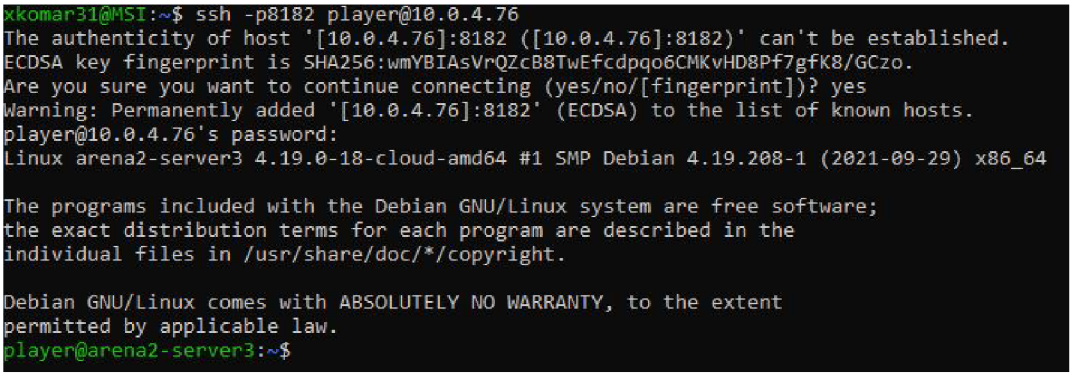
Po zadání příkazů:

```
ssh -p4350 player@10.0.4.76
ssh -p8182 player@10.0.4.76
```

Úspěšné tunelování k virtuálnímu serveru3 v prvním scénáři je možné vidět na obrázku 3.6.


Obr. 3.6: Připojení přes Jump server do 1. scénáře serveru-3.

V případě druhém, je možné vidět úspěšné připojení k serveru3 druhého scénáře zde 3.7.




Obr. 3.7: Připojení přes Jump server do 2. scénáře serveru-3.

Všechny scénáře včetně Jump serveru jsou smazány po zadaní příkazu:

```
ansible-playbook site.yaml -e operation=game_shutdown
```

### 3.3.4 Tunelování přes OpenStack funkci

V druhém případě spuštění aplikace s módem floating_ip, jsou taktéž vygenerovány scénáře s Jump serverem i s Jump sití. I když je vytvořen Jump server s Jump sití je využívána jen Jump síť. Server3 z prvního i druhého scénáře jsou připojeny do Jump sítě a následně obdrží plovoucí IP adresy. Obdržené IP adresy jsou následně využívány k připojení k Instanci z fyzické sítě. Získané veřejné IP adresy jsou vyznačené na obrázku 3.8.

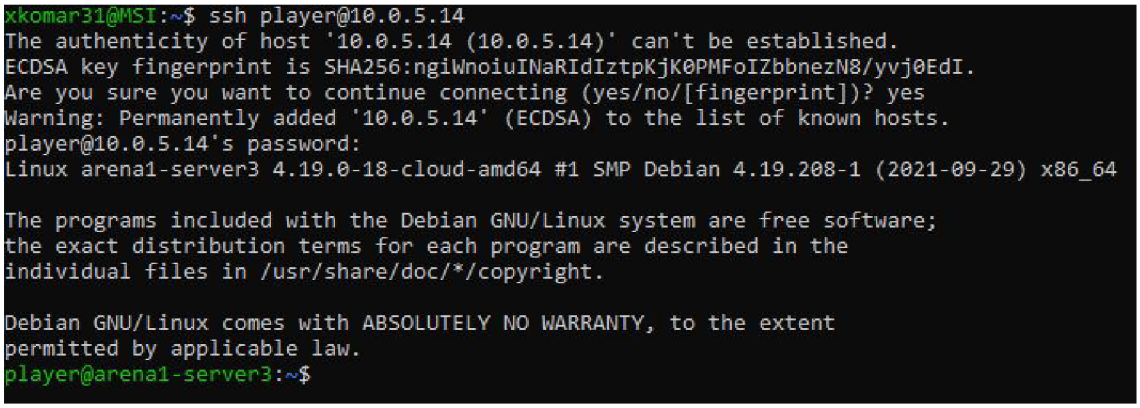
| Instance Name | Image Name | IP Address |
|---|---|---|
| Arena2-server3 | debian-10 | **Arena-Jump-Network** 192.168.100.245, 10.0.4.41<br>**Arena2-net2** 192.168.20.13 |
| Arena2-server2 | debian-10 | 192.168.10.12 |
| Arena2-server1 | debian-10 | 192.168.10.11 |
| Arena1-server3 | debian-10 | **Arena1-net2** 192.168.20.13<br>**Arena-Jump-Network** 192.168.100.7, 10.0.5.14 |
| Arena1-server2 | debian-10 | 192.168.10.12 |
| Arena1-server1 | debian-10 | 192.168.10.11 |

Obr. 3.8: Vyznačené plovoucí IP adresy.

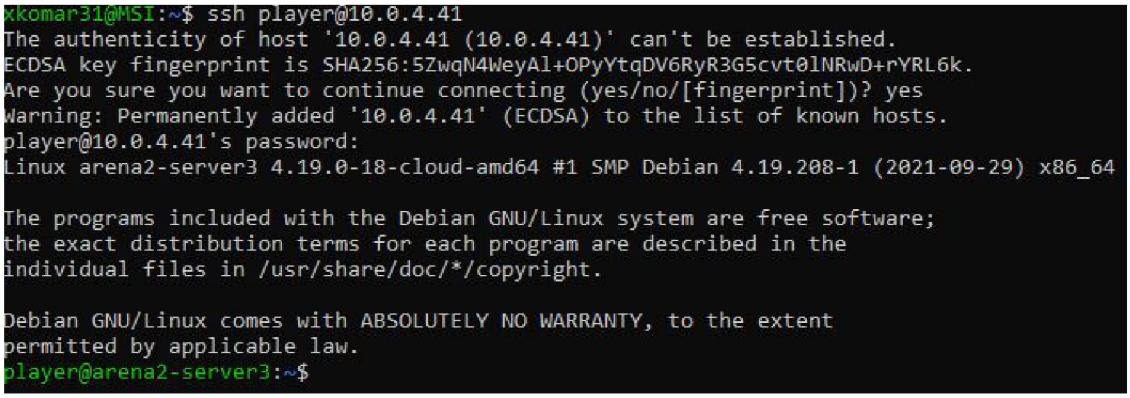
Příkazy k připojení virtuálním serverům přes plovoucí adresu jsou následující:

```
ssh player@10.0.5.14
ssh player@10.0.4.41
```

Po zadání příkazu je ještě nutné zadat vygenerovaná hesla pro servery. Úspěšné připojení k serveru3 z prvního scénáře je možné vidět na obrázku 3.9 a k serveru z druhého scénáře na obrázku 3.10.




Obr. 3.9: Připojení do 1. scénáře serveru3.




Obr. 3.10: Připojení do 2. scénáře serveru3.

# Závěr

Cílem práce bylo navrhnout implementaci Jump serveru pro tvorbu SSH tunelů ke vzdáleném virtuálním strojům z fyzické sítě. Před samotnou implementací autor práce seznamuje čtenáře s problematikou virtualizace a kontejnerizace. Jelikož implementace je pro cloudové prostředí OpenStack, tak se čtenář seznamuje i se základními pojmy a službami ohledně OpenStacku. Jakými službami je toto prostředí tvořeno. V rámci jednotlivých služeb jsou popsány postupy k jejich využití, například jak vytvořit instanci, síť. Na závěr teoretické části proběhlo porovnávání VMs a kontejnerů, kde na základě výsledků porovnání VMs a kontejnerů je vybrána technologie v podobě kontejneru.

Praktická část se skládá ze dvou kapitol. V první kapitole je prováděna manuální implementace Jump serveru. Před samotnou implementací je také prováděn výzkum ohledně způsobu zvolení technologie tunelování a způsobu implementace Jump serveru. Na základě výsledku manuální implementace kontejneru je prováděna i implementace Jump serveru jako virtuálního stroje. Z důvodu komplikací při kontejnerizaci Jump serveru a úspěšného výsledku při virtualizi Jump serveru, byla prováděna automatizace virtualizací. Hlavní problém při kontejnerizaci nastával v nastavení síťování v kontejneru.

V druhé části proběhla automatizovaná implementace Jump serveru do scénářů, které budou generovány v rámci Kybernetické arény. Jump server byl automatizován jako virtuální stroj a pro SSH tunelování byl využit Nginx server. Automatizace je prováděna přes Ansible server, kde je nainstalován automatizační nástroj Ansible. Na základě potřebných procesů k orchestraci Jump serveru a scénářů, byly vytvářeny potřebné automatizační role s playbooky.

K vytváření orchestrační aplikaci bylo nutné se seznámit s automatizačním nástrojem Ansible. Také se zorientovat v obdržených automatizačních souborech a vymyslet automatizační strukturu pro Jump server a jeho funkcí. Jakým způsobem aplikace funguje a je implementována je popsáno v rámci druhé kapitoly praktické části, kde je prováděna automatizace a seznámení čtenáře funkcionalitou aplikace.

# 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 |