

UNIVERZITA HRADEC KRÁLOVÉ
FAKULTA INFORMATIKY A MANAGEMENTU
KATEDRA INFORMATIKY A KVANTITATIVNÍCH METOD

Orchestrace a management virtuálních
síťových funkcí

DIPLOMOVÁ PRÁCE

Autor: Bc. Ondřej Smola

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Vladimír Soběslav, Ph.D.

Hradec Králové

srpen, 2016

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a uvedl jsem všechny použité prameny a literaturu.

V Hradci Králové dne 29. srpna 2016

Ondřej Smola

Poděkování

Děkuji vedoucímu diplomové práce, Ing. Vladimíru Soběslavovi, Ph.D, za metodické vedení práce, odborné rady a připomínky v průběhu jejího psaní. Dále bych chtěl poděkovat za podporu své rodině, kolegům a přátelům.

Anotace

Tato diplomová práce se zaměřuje na problematiku spojenou s virtualizací síťových funkcí (NFV). Jedná se velice aktuální a dynamickou oblast, která si klade za cíle transformovat síťovou funkcionalitu z hardwarových prvků do softwarových aplikací či virtuálních instancí. Ty následně mohou těžit z výhod cloudových platforem. Hlavním cílem této práce je popsat oblast NFV, se zaměřením přímo na virtuální síťové funkce (VNF). Na závěr této práce jsou získané poznatky využity pro vytvoření ukázkových příkladů pro VNF, která mohou být využita na cloudové platformě.

Annotation

This master thesis focuses on network function virtualization (NFV). It's a very current and dynamic field, which goal is to transform network functionality from hardware appliances to software applications or virtual machines. They can then profit from benefits of cloud platforms. Main goal of this thesis is to describe field of NFV with direct focus on virtual network function (VNF). At the conclusion of this work are learned knowledge used to create a sample examples of VNF, which can be used to cloud platform.

Obsah

1. Úvod	1
2. Základní problematika virtualizace síťových funkcí	3
2.1. Virtualizace a Cloud Computing	3
2.2. Potřeba virtualizace síťových funkcí	5
2.3. NFV a softwarově definované sítě (SDN)	7
2.3.1. Service Chaining	9
3. Dostupné technologie NFV a VNF	12
3.1. Architektura NFV a VNF	12
3.1.1. Infrastruktura NFV	13
3.1.2. Virtuální síťová funkce	15
3.1.3. Management a orchestrace NFV	16
3.2. Dostupné prostředky pro NFV Infrastrukturu	18
3.2.1. OpenStack	18
3.2.1.1. Vanilla Neutron	20
3.2.1.2. OpenContrail	21
3.2.2. VMware vCloud Suite	21
3.3. Dostupné možnosti pro VNF	23
3.3.1. Firewall as a Service	24
3.3.2. Load balancer as a Service	24
3.4. Možnosti prostředky pro Management a Orchestraci VNF	25
4. Požadavky a návrh prostředí pro NVF a VFN	28
4.1. Požadavky a výběr platformy pro NFV Infrastrukturu	28
4.2. Požadavky a výběr řešení pro VNF	30

4.3. Výsledná architektura použitého frameworku	31
5. Testovací scénáře a realizace pro VNF a NFV	33
5.1. Scénáře pro použití vybraných VNF	33
5.1.1. Scénář LbaaS	33
5.1.2. Scénář FwaaS	35
5.2. Realizace VNF pro LbaaS	36
5.2.1. HAproxy - Neutron HAproxy agent	36
5.2.1.1. Heat template pro haproxy	37
5.2.1.2. Testování heat templatu pro haproxy	40
5.2.2. AVI networks	41
5.2.2.1. Heat template pro AVI networks	42
5.2.2.2. Testování heat templatu pro AVI networks	45
5.3. Realizace VNF pro FwaaS	45
5.3.1. Servisní instance v OpenContrailu	45
5.3.2. Heat template pro FwaaS	48
5.3.3. Management PfSense	52
5.3.4. Management Fortigate VM	54
5.4. Shrnutí získaných poznatků a zhodnocení	56
6. Závěr	58
 Literatura	 I
 Přílohy	 XII
A. Heat template Lbaas - HAproxy	XIII
B. Enviroment file pro LbaaS - HAproxy	XX
C. Heat template FwaaS	XXI
D. Enviroment file pro LbaaS - Pfsense	XXIX
E. Enviroment file pro LbaaS - Fortigate VM	XXXI

1. Úvod

V dnešní době dochází v datových centrech k nasazování nových moderních technologií. V oblasti výpočetního výkonu a úložišť se jedná především o virtualizaci a cloud computing. Jak například udává [1], tak v době psaní této práce 95% IT profesionálů používá nějaký typ cloudové platformy. Přechází se tedy z hardwareově orientovaných data center na virtuální cloudová data centra. Je již tedy běžnou praxí, že v datových centrech vše běží na rozsáhlé fyzické infrastruktuře, která je abstrahovaná na jeden souvislý blok výpočetního výkonu a jeden souvislý blok úložiště.

Dalším takovýmto funkčním blokem, který je součástí datových centrech a je velice důležitou součástí infrastruktury velkých společností, jsou počítačové sítě. V oblasti počítačových sítí byl, oproti dvěma zmíněným oblastem, pomalejší vývoj inovací. Je to z důvodu toho, že počítačové sítě jsou velmi komplexní oblastí a také to, že produkční vývoj v telekomunikačním průmyslu se tradičně řídil přísnými standardy kvůli stabilitě a kvalitě komunikace, jak zmiňuje [2]. Přestože tento model v minulosti fungoval, tak vedl nevyhnutelně k dlouhým produkčním cyklům, pomalému tempu vývoje a spoléhání se na proprietární či specializovaný hardware.

Avšak i zde je snaha změnit dosavadní návrh a fungování počítačových sítí. Je zde snaha využívat nové přístupy a technologie, které umožní flexibilní a rychlé nasazování nových síťových služeb a zároveň snížit jejich náklady. Jedná se především o využití již zmíněné virtualizace a programatické správy sítě. [3]

Jedním z nových přístupů je virtualizace síťových funkcí (Network functions virtualization - NFV), kterou poprvé navrhl ETSI v publici [4]. Virtualizace síťových funkcí se zaměřuje na transformaci způsobu, jakým síťový architekti při-

stupují k oblasti počítačových sítí především u telekomunikačním poskytovatelů služem. Snaha je tedy přesunout mnoho typů síťového příslušenství z fyzických síťových prvků do standardních průmyslově používaných serverů a úložišť, které mohou být umístěny v datových centrech či přímo u koncových zákazníků. Tímto lze dosáhnout virtuálních síťových funkcí, které mají naprosto stejnou funkcionality jako síťové funkce umístěné v síťových prvcích, avšak získávají výhody spojené s virtualizací a cloud computingem. NFV je relativně nová oblast, ve které je mnoho prostoru pro inovace a zlepšení, jak popisují [5] a [6]. Jedním z nich je i management a orchestrace virtuálních síťových funkcí, kterou se zabývá tato práce.

Cílem této diplomové práce je analyzovat a navrhnout řešení pro management a orchestraci jednoduchých virtuálních síťových funkcí, které by mohli využít uživatelé cloudové platformy. Celé řešení spočívá v návržení architektury pro NFV frameworku, na kterém následně bude provedeno testování několika virtuálních síťových funkcí a jejich následné porovnání a zhodnocení. Výsledná řešení, které budou výstupem této práce, by měla sloužit jako obecný blueprint pro tvorbu NFV a VNF v cloudových prostředí. V celé práci budou využívány pouze aktuálně dostupné technologie, které by následně mohli být využity i produkčním prostředím.

Celá struktura této práce je rozdělena na několik částí. V druhé kapitole jsou vysvětleny hlavní pojmy a problematika oblasti virtualizace síťových funkcí. Třetí se zabývá referenční architekturou NFV a popisem možných technologií. Ve čtvrté kapitole je popsána již výsledná architektura a použité technologie společné s odůvodněním pro jejich vybrání. Pátá kapitola je následně věnována testování a realizaci jednotlivých virtuálních síťových funkcí. Na konci této práce je poté uvedeno závěrečné shrnutí.

Závěrečná práce byla zpracována ve spolupráci s firmou tcp cloud a.s., která poskytuje implementace jednoho z nejlepších cloudových řešení na světě. Firma umožnila využít jejich stávající infrastrukturu v nejmodernějším datovém centru v České republice, které je v budově Technologického centra Písek s.r.o.

2. Základní problematika virtualizace síťových funkcí

Tato kapitola se zabývá základní analýzou a popisem problematiky spojené s oblastí virtuální síťových funkcí. Nejprve uveden přehled a krátký popis virtualizace a cloud computingu spolu s jejich přínosem pro IT. Následně je vysvětlena potřeba virtualizace síťových funkcí. Zde jsou porovnány jednotlivé přístupy k řešení a návrhu počítačových sítí. Zároveň je zde vysvětlen i základní koncept virtualizace síťových funkcí.

2.1. Virtualizace a Cloud Computing

Virtualizace je technologie, která změnila způsob, jakým se přistupuje k IT infrastruktuře. V tradičním modelu IT infrastruktury servery podporují pouze jeden operační systém v daném čase. Na tomto systému obvykle běží pouze jedna aplikace. Přestože by na tomto systému mohlo běžet více aplikací, tak je lepší držet aplikace odděleně na různých systémech z důvodu minimalizace potenciálních bodů selhání. Pokud například nastane s aplikací problém, tak častým řešením je restartování systému. Pokud by na systému bylo více aplikací, znamenalo by to jejich vyřazení z provozu po dobu restartu, který může trvat velice dlouho. [7]

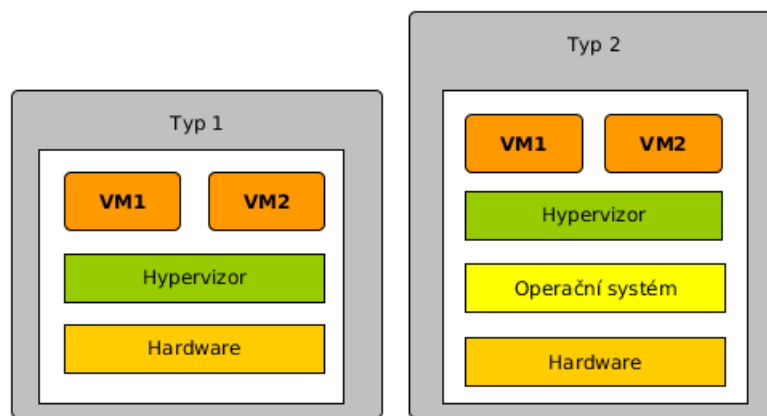
Z výše zmíněných důvodů došlo k velkému rozvoji a nasazování virtualizace. Virtualizací obecně označujeme techniky, které umožňují k dostupným hardwarovým zdrojům přistupovat jiným způsobem, než jakým fyzicky existují. Je tomu díky softwaru, který tento hardware abstrahuje a vytvoří tím virtuální prostředí. Virtualizované prostředí se dá snadněji přizpůsobit potřebám uživatelů, případně

skrýt pro uživatele nepodstatné detaily (jako např. rozmístění hardwarových prostředků). Tím je tedy umožněno na jednom fyzickém serveru provozovat více od sebe oddělených virtuálních strojů, které mají každý svůj vlastní operační systém s aplikacemi.

Software, který slouží pro virtualizaci, se nazývá hypervisor. Jak zmiňuje Popel a Goldberg v [8], tak existují tyto dva základní typy hypervisorů:

- Typ 1 (Nativní - Bare-metal) - Tento hypervisor běží přímo na fyzickém hardwaru. Tím umožňuje provozovat více operačních systémů na jednom fyzickém stroji. Příkladem takového hypervisoru je VMware ESXi, XEN a KVM.
- Typ 2 (Hostovaný) - Na rozdíl od předchozího případu tento typ hypervisoru běží v prostředí operačního systému. Příkladem je například velice oblíbený Virtualbox či VMware Workstation.

Obrázek 2.1 zobrazuje schématický popis obou typů hypervisorů a jejich rozdíl.



Obrázek 2.1.: Schéma hypervisorů

Virtualizace je základní technologie, na které je postavena virtualizace síťových funkcí. Ve většině případů je však využití pouze jednoho hypervisoru telekomunikačními provozovateli a velkými společnostmi nedostačující a je proto nutné využít cloud computingu.

Existuje několik definic pro cloud computing. Nejčastější používaná definice pro cloud computing je [9], kde je uvedeno, že je to model umožňující využívání

společného poolu počítačových zdrojů vzdáleně přes počítačovou síť. Tyto zdroje mohou být flexibilně alokovány a uvolňovány dle potřeby. Základní charakteristiky cloudu jsou tedy:

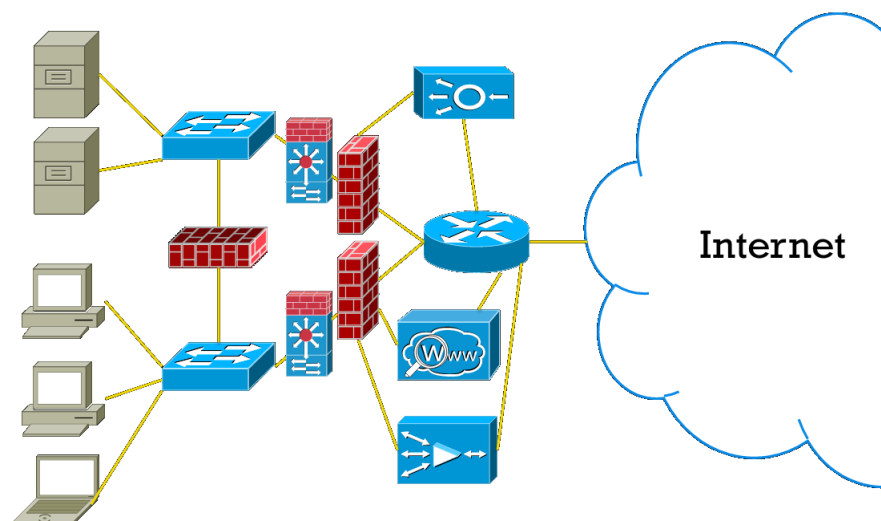
- Služby dostupné na požádání
- Všudypřítomný přístup k síti
- Sdílení zdrojů
- Vysokou elasticitu a flexibilitu
- Měření využitých zdrojů

Z technického hlediska je cloud resp. cloudová infrastruktura několik společně propojených serverů v clusteru. Na těchto serverech běží hypervisor, který vytvoří virtuální infrastrukturu. Tímto způsobem je tvořen společný pool zdrojů. Pro vytváření cloudových služeb zde ještě musí existovat cloudová platforma, která dokáže celou tuto virtuální infrastrukturu centrálně spravovat. Výhodou tohoto přístupu je efektivní využívání dostupných výpočetních zdrojů a velká flexibilita v poskytování a využívání služeb. Více informací lze najít například v [10].

2.2. Potřeba virtualizace síťových funkcí

Tradiční počítačové sítě jsou složeny ze značného množství propojených hardwarových zařízení, jako jsou např. routery, switche a firewally. Tyto zařízení jsou speciálně určena pro daný účel a jejich řízení může být zkrze distribuované protokoly. Tím se počítačové sítě stali stabilními a vysoce výkonnými, avšak omezilo je to ve flexibilitě. Tradiční schéma sítě je zobrazeno na obrázku č. 2.2.

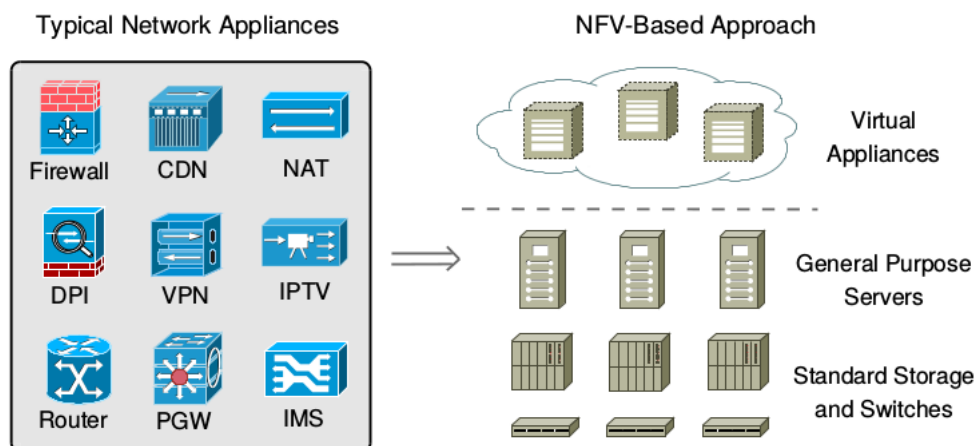
Z obrázku je patrné, že takovéto sítě se mohou skládat z desítek či i stovek hardwarových boxů. Management je tedy velice náročný a velice často může docházet chybám v důsledku lidské chyby. Bylo tedy nalézt lepší řešení. V [11] je například navrženo přesunout většinu funkcionality do veřejného cloudu a tím správu sítě



Obrázek 2.2.: Tradiční schéma počítačové sítě

přenechat třetí straně. To však pro telekomunikační poskytovatele a většinu velkých firem není vhodné řešení.

Z tohoto důvodu bylo navrženo koncept NFV. Celá myšlenka NFV je založena na tom, že dojde k separování softwarové funkcionality v síťových prvcích od proprietárního hardwaru, na kterém běží. Jak znázorňuje obrázek č. 2.3.



Obrázek 2.3.: Koncept virtualizace síťových funkcí, převzato z [5]

To umožní se síťovými funkcemi zacházet jako s klasickými softwarovými aplikacemi, které mohou běžet na standardním komerčně dostupných serverech jenž organizace v současnosti používají. Díky tomu může být také využito cloudu (pri-

vátních) a virtualizace. To umožní zlepšit následující aspekty provozu telekomunikačních sítí.

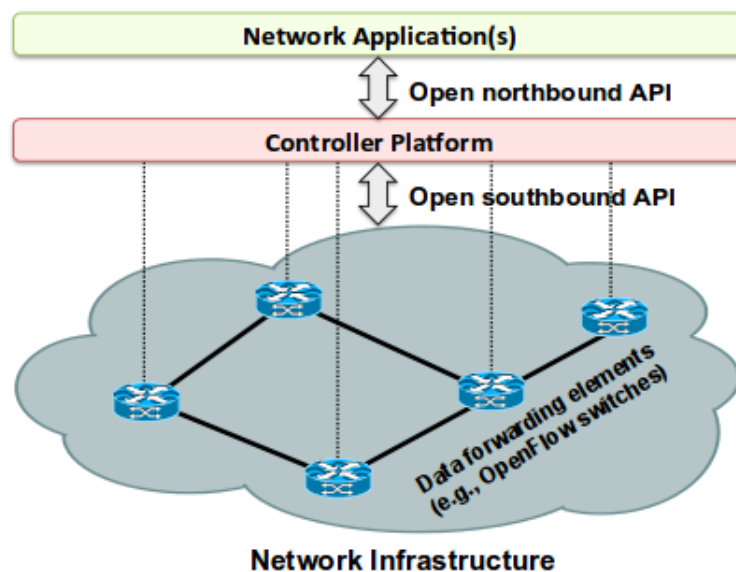
- Smíření investičních nákladů – snížení potřeby nákupu jednoúčelových hardwarových zařízení, možnost platby pouze za využití kapacity a snížení rizik přílišného předimenzování kapacit
- Snížení provozních nákladů – snížení prostoru, napájení a požadavky na chlazení, zjednodušení správy a řízení síťových služeb
- Urychlení Time-to-market – zkrácení doby pro nasazení nových síťových služeb, chopení se nových příležitosti na trhu, vyhovění potřebám zákazníka
- Doručit agilitu a flexibilitu – možnost rychle škálovat (rozšiřovat nebo zmenšovat služby) dle měnících se požadavků od zákazníka. Podpora služeb, které mají být dodány pomocí softwaru na libovolném standardním serverovém hardwaru

Za zmínění stojí poznámka v [6], kde je řečeno, že obecný koncept oddělení síťové funkce od hardwaru ještě nutně neznamena potřebu využití virtualizace. Protože budou síťové funkce dostupné jako software, tak mohou být nainstalovány a provozovány přímo na fyzickém stroji. Ovšem rozdíl je, že tento stroj již nebude speciální hardware, ale klasický server. Tento scénář může být do jisté míry použit při nasazování síťových funkcí v malém měřítku např. v uživatelských koncových bodech. Avšak pro plné využití všech výše zmíněných výhod, které jsou třeba ve velkých datových centrech, je třeba s použitím virtualizace počítat.

2.3. NFV a softwarově definované sítě (SDN)

Kromě NFV existuje i další technologie, která se snaží zlepšit řízení a správu počítačových sítí. Jde o softwarově definované sítě (SDN). Dle [12] jde o koncept, ve které je oddělena řídicí logika (control plane) z jednotlivých routerů a switchů, které přeposílají traffic (data plane). Tím, že dojde k oddělení datové a řídicí vrstvy,

se routery a switche stanou pouze přeposílající data a veškerá řídicí logika může být implementována v jednom logicky centrálním místě (SDN Controller). Z tohoto centrálního místa lze do jednotlivých routerů a switchů předávat instrukce pomocí aplikačních programovacích rozhraní (API). Samotný SDN Controller také obsahuje API, které mohou využívat aplikace a tím řídit, resp. programovat celou počítačovou síť.



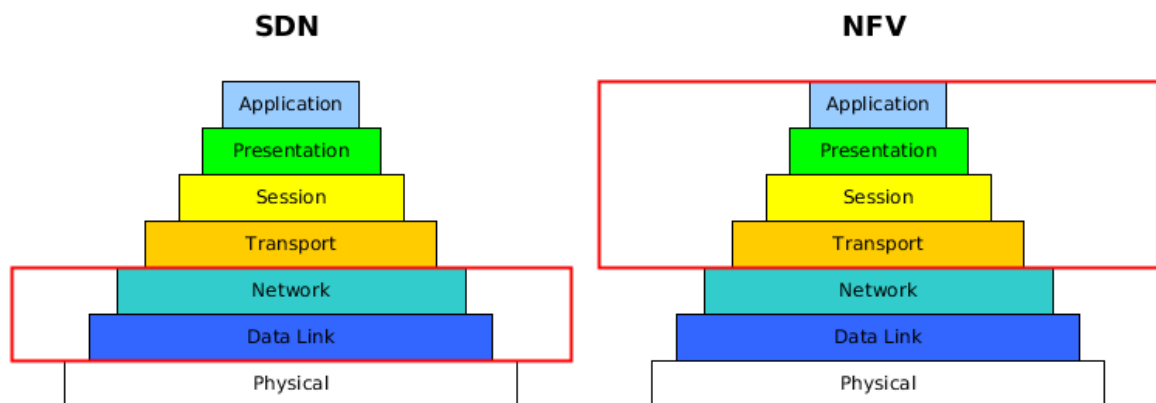
Obrázek 2.4.: Schéma SDN, převzato z [12]

Obrázek č. 2.4 ukazuje jednoduché schéma softwarově definovaných sítí. Jak je z obrázku patrné, tak celou architekturu lze tedy rozdělit do 3 logických vrstev, které spolu komunikují pomocí API.

- **Aplikační vrstva** - Na této úrovni se nachází samotné síťové aplikace jako jsou například DHCP, ACL, NAT, DNS a další. Jejich vytváření by mělo být poskytováno prostřednictvím nižší vrstvy, nazývané northbound API.
- **Control vrstva** - V této vrstvě je centralizována veškerá logika, které dříve byla v síťových prvcích.
- **Vrstva infrastruktura** - Nejnižší vrstvou je samotný hardware pro předávání datagramů na fyzické úrovni. Pro funkčnost celé architektury je nutné, aby

zde byla nasazena zařízení, která umí přijímat pokyny od control plane skrze southbound API.

Přestože Softwarově definované sítě a virtualizace síťových funkcí jsou dvě různé technologie a koncepty, tak dle [13] využití obou těchto technologií je nejlepší cestou pro telekomunikační poskytovatelé a velké firmy. Je to z důvodu toho, že se každá orientuje na jinou část datového provozu. To znázorněno na obrázku č. 2.5.



Obrázek 2.5.: Srovnání SDN a NFV

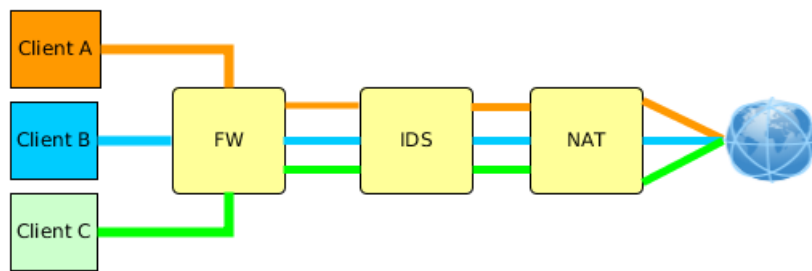
Fakt je, že SDN umožňuje programaticky ovládat počítačovou síť. To lze využít u NFV pro poskytnutí programovatelné konektivity mezi jednotlivými virtuálními síťovými funkcemi. Tento proces nazývá Service Chaining.

2.3.1. Service Chaining

Jednou z výhod NFV a SDN je možnost využít Service Chaining. Service chaining je ve skutečnosti součástí SDN. Jde o princip jakým lze dynamicky pospojovat jednotlivé VNF a ovládat tak toky v síti. [14]

Service chaining není ve skutečnosti nic nového. V klasických počítačových sítích je používán také, ale pomocí fyzických síťových prvků. Jedná se zjednodušeně o způsob zapojení mezi jednotlivými síťovými prvky (či VNF) a způsob, jakým na sebe navazují. Příklad takového zapojení je vidět na obrázku č. 2.6.

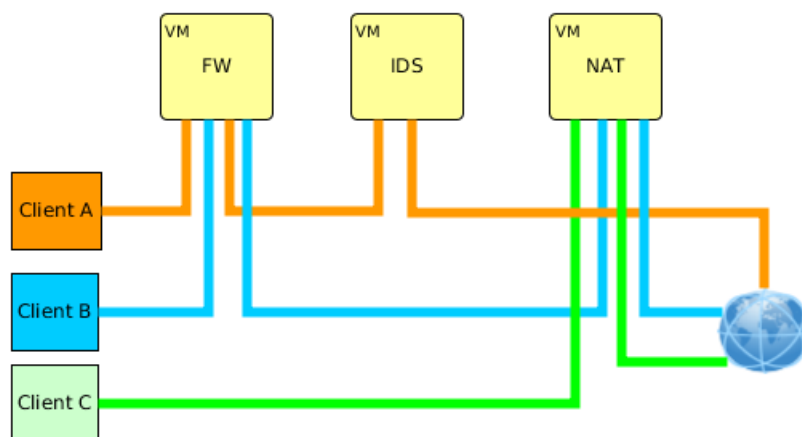
Zde se provozovatel sítě rozhodl, že odchozí data z klientských stanic musí jít přes firewall, IDS a nakonec přes NAT do Internetu. Příchozí data mají logicky



Obrázek 2.6.: Ukázka klasického service chainigu pomocí fyzických síťových prvků

obrácené pořadí. Toto zapojení funguje dobře pro síť, kde není třeba rozlišovat cestu jakou proudí data jednotlivých uživatelských stanic. Ale není to optimální řešení pro síť s více uživateli, kde každý požaduje jinou síťovou funkci. Také je možné, že se potřeby pro jednotlivé síťové služby mohou měnit v čase. Příklad takové sítě lze nalézt ve většině datových center.

Zde tedy přichází na řadu VNF spolu s SDN. Protože jednotlivé VNF existují jako virtuální stroje, tak mohou být dynamicky nasazovány dle aktuálním požadavků jednotlivých klientů a pomocí SDN mohou být tyto VM dynamicky pospojovány. Obrázek č. 2.7 ukazuje schéma zapojení, kde každý klient může mít jinou požadovanou cestu do internetu. Je možná i varianta, kde každý klient má své vlastní VNF s jinou konfigurací.



Obrázek 2.7.: Ukázka VNF service chainigu

Je tedy vidět, že NFV a SDN jdou velice dobře dohromady a lze pomocí nich

získat mnoho výhod. Vzhledem k tomu, že tato práce se zabývá managementem a orchestrací VNF, tak zde tedy nutné počítat s využitím SDN při návrhu NFV frameworku, na kterém budou následně testovány jednotlivé VNF.

3. Dostupné technologie NFV a VNF

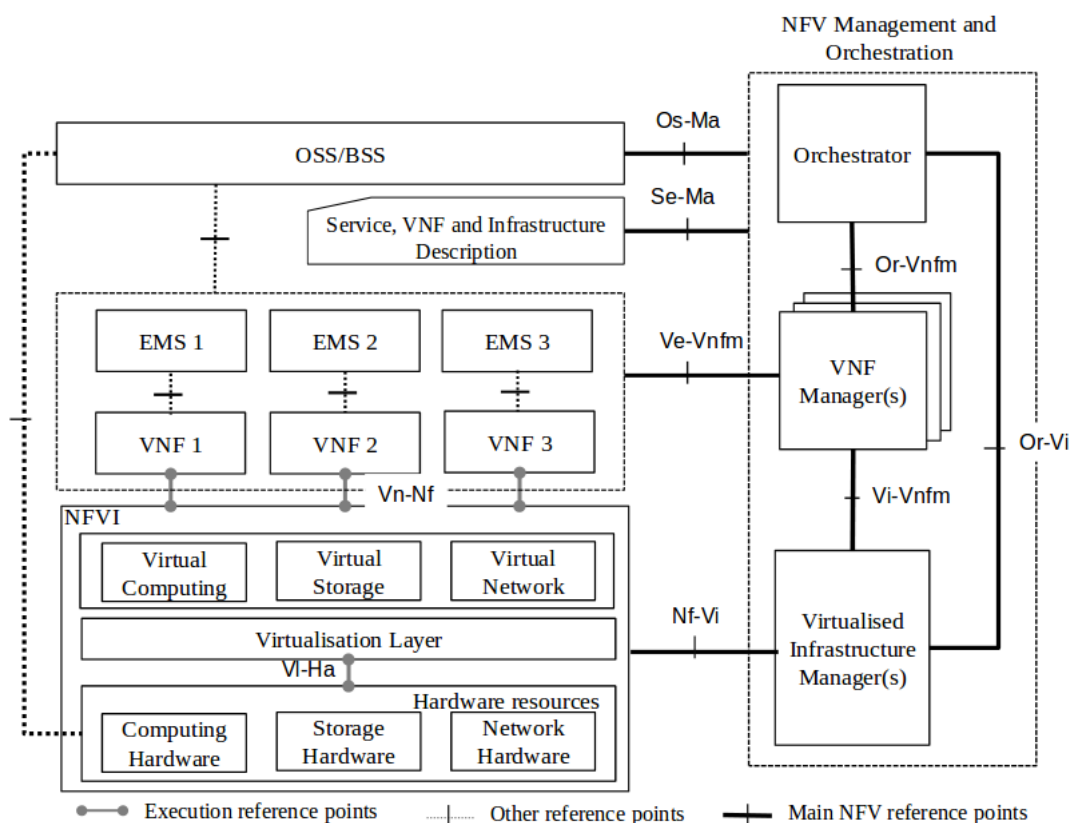
V předchozí sekci byla popsána myšlenka a motivace související s virtualizací síťových funkcí. V této kapitole bude probrána architektura pro NFV a následně možnosti pro implementaci jednotlivých částí této architektury.

3.1. Architektura NFV a VNF

V [15] je popsána referenční architektura pro NFV, která byla navržena organizací ETSI. Jedná se pouze o funkční návrh bez náznaků konkrétní implementace. Obrázek č. 3.1 znázorňuje tuto architekturu.

Z obrázku je patrné, že celá architektura se dá rozdělit na tyto 3 hlavní části, které mezi sebou mají komunikovat. Tyto části jsou:

- **Infrastruktura virtualizace síťových funkcí (NFVI)** - Jsou všechny softwarové a hardwarové zdroje potřebné k vytvoření prostředí, ve které mohou být jednotlivé VNF být nasazeny. Tato infrastruktura může být velice rozsáhlá, proto je její součástí i síť poskytující konektivitu mezi vzdálenými lokacemi infrastruktury.
- **Virtualizované síťové funkce (VNFs)** - Jsou softwarové implementace síťových funkcí, jako je např. NAT a routing, které mohou být nasazeny na NFV infrastruktuře.
- **Management a orchestrace NFV (NFV-MANO)** - zde se jedná o řízení softwarových a hardwarových zdrojů v celé infrastruktuře NFV a životního cyklu jednotlivých virtuálních síťových funkcí. Tato část se tedy zaměřuje na řízení a správu všech úloh související v virtualizaci v NFV frameworku.



Obrázek 3.1.: NFV architektura, převzato z [15]

Podrobné vysvětlení všech zkratk a terminologii, která je vyobrazena na obrázku lze nalézt v [16]. Pro účely této práce jsou dále popsány pouze výše uvedené hlavní části architektury.

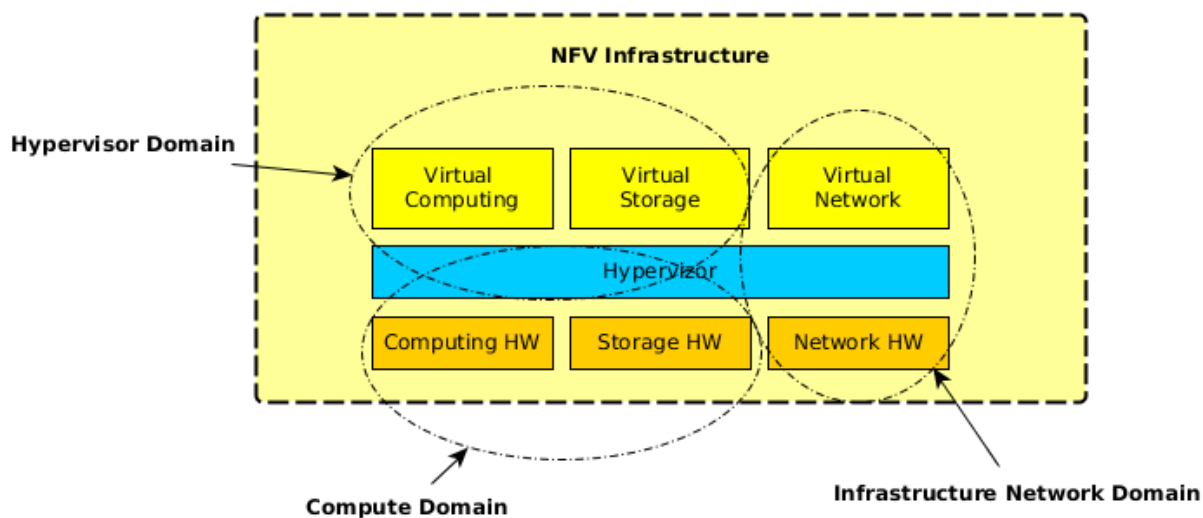
3.1.1. Infrastruktura NFV

Ve zdroji [17], který detailně popisuje infrastrukturu pro virtualizaci síťových funkcí (NFVI), je uvedeno, že je v ní sdružení všech základních zdrojů potřebných pro běh virtuálních síťových funkcí (VNF). Z tohoto důvodu sem patří veškerý hardware. Do NFVI také patří některé softwarové komponenty, které jsou společné pro mnoho VNF a poskytují funkcionalitu potřebnou pro podporu nasazení, propojení či managementu VNF. Celou infrastrukturu může tvořit jeden či více strojů, které mají tyto potřebné funkce. Tyto stroje také mohou být umístěny v různých spolu spojených geografických lokacích.

Pro zjednodušení lze celou NFV infrastrukturu rozdělit do 3 následujících domén:

- **Compute Domain** - Do této domény patří veškeré hardwarové zdroje jako jsou servery, úložiště a komponenty, které tyto zdroje obsahují, např. procesory, pevné disky, síťové karty, atd. Zároveň je zde řešen návrh fyzické topologie. [18]
- **Hypervisor Domain** - Toto je doména, které představuje softwarové prostředí abstrahující hardware v compute doméně a poskytuje je jako virtuální zdroje. Tyto zdroje následně mohou využívat virtuální síťové funkce. [19]
- **Infrastructure Network Domain** - V této doméně je řešeno veškeré propojení výše zmíněných domén. Tedy fyzické i virtuální infrastruktury. [20]

Funkci obsaženou v jednotlivých doménách znázorňuje obrázek č. 3.2. Více informací na tuto problematiku lze nalézt v [17] a ve zdrojích uvedených u každé domény.



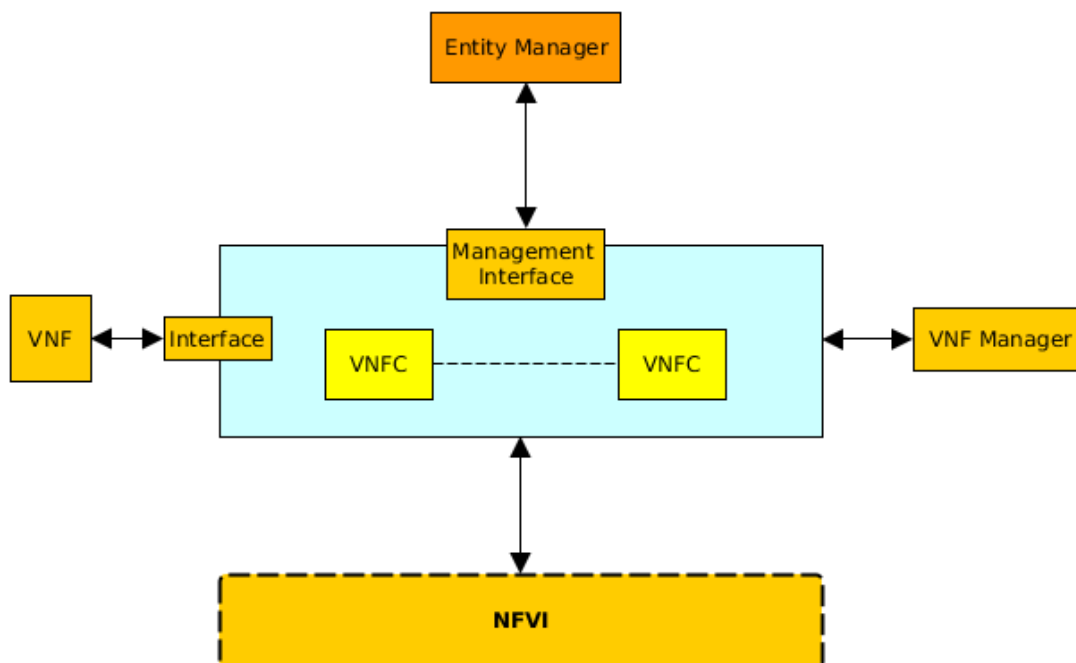
Obrázek 3.2.: Schéma NFV infrastruktury

Dá se říci, že referenční návrh infrastruktury pro NFV je podobný jako pro návrh infrastruktury pro cloud computing platformu. V kapitole 3.2 jsou uvedeny příklady možných cloudových platform, které lze NFVI využít.

3.1.2. Virtuální síťová funkce

Virtuální síťová funkce (VNF) je dle [21] určitá síťová funkce, která běží na NFV infrastruktuře a je zároveň NFV frameworkem řízena a spravována. Zároveň musí mít dobře definované rozhraní k ostatním síťovým funkcím, k VNF Managerovi a měla by obsahovat management rozhraní či port. Jedna VNF může být obsažena v jednom virtuálním stroji nebo může být roztažena přes více virtuálních strojů.

Na obrázku č. 3.3 je vidět jednoduché schéma virtuální síťové funkce dle referenčního návrhu [21]. Celý životní cyklus VNF (vytvoření, spuštění, zastavení, smazání a škálování) řídí VNF Manager, který je součástí NFV managementu a orchestrace. Současně je možné dynamicky změnit aktuální konfiguraci pomocí Entity manageru (EM) přes management interface. EM může spravovat více VNF nebo právě jednu. Vnitřní struktura celé instance může být tvořena více komponentami (VNFC), které spolu mohou být navzájem provázány. Toto provázání však nemusí být viditelné zvenčí.



Obrázek 3.3.: Schéma virtuální síťové funkce

Pohledem na současný trh zjistíme, že VNF je prakticky poskytována ve 3 základních podobách.

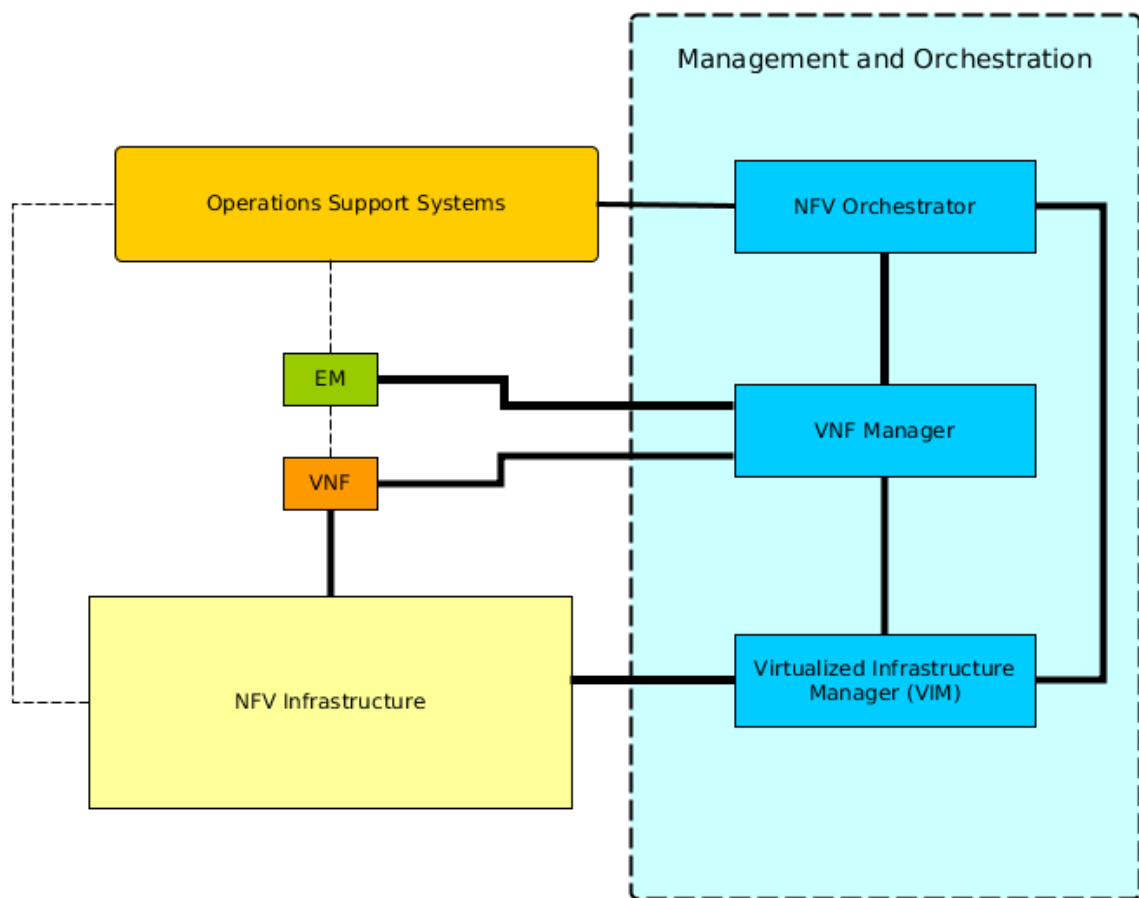
- Softwarová aplikace - V tomto případě je poskytována VNF jako aplikace, která může být nainstalována na běžný operační systém jako je například GNU/Linux.
- Ucelený operační systém - Zde je poskytován přímo celý operační systém, který může být nainstalován do virtuálního stroje nebo i na fyzický server.
- Kompletní VM - Poskytovatel VNF může dát k dispozici rovnou přetvořený obraz virtuálního stroje (image), který může obsahovat operační systém se síťovými funkcemi. Tento systém však nemusí být klasicky dostupný operační systém jako je GNU/Linux či FreeBSD, ale může se jednat o speciálně vytvořený systém od výrobce. Tento způsob budou využívat poskytovatelé, kteří mají proprietární řešení pro síťová řešení jako je například Cisco či Juniper.

V kapitole 3.3 jsou uvedeny příklady softwaru existujícím na současném trhu, který lze využít jako VNF.

3.1.3. Management a orchestrace NFV

Management a orchestrace virtualizace síťových funkcí (NFV MANO) je nejdůležitější část celého NFV frameworku. Je tomu tak, protože MANO zajišťuje správné fungování NFV infrastruktury i jednotlivých virtuálních síťových funkcí. MANO také poskytuje funkce nutné pro provisioning VNF a související operace, jako je jejich konfigurace jednotlivých VNF a infrastruktury, na které běží. Zároveň spravuje a řídí životní cyklus fyzických a virtuálních zdrojů, které slouží pro podporu VNF.

Jak vyplývá z obrázku č. 3.4, tak referenční návrh MANO dle [22] se skládá ze hlavních 3 částí, které se zabývají správnou jednotlivých vrstev NFV frameworku.



Obrázek 3.4.: Schéma NFV MANO

- Virtualized infrastructure manager (VIM) - Řídí a spravuje fyzické a virtuální zdroje v jedné doméně infrastruktury. Celková infrastruktura se může skládat z více domén a každá musí mít svůj VIM. Jeho typickými úlohami jsou vytváření, udržování a uvolňování VM na dostupných zdrojích v doméně. Zároveň musí mít přehled o všech těchto a stavu hardwarových zdrojů.
- VNF manager - Dohlíží na lifecycle management jednotlivých VNF instancí. To znamená, že vytváří, udržuje a ukončuje VNF instance, které běží na jednotlivých VM (ty však spravuje VIM). Opět může existovat více VNF managerů, kteří mohou spravovat jednu či více VNF.
- NFV orchestrator - Zjednodušeně slouží jako řízení a správu všech VIM a všech VNF managerů. Pomocí komunikace s VIM dokáže spravovat dostupné

zdroje a pomocí komunikace s VNF managery dokáže řídit síťové služby. Jeho další funkcí je i přehled všech dostupných VNF, neboli katalog VNF, a registrace nových VNF do tohoto katalogu. Ten je pak dostupný uživatelům.

Celý systém je navržen tak, že by měl pracovat společně se stávajícími aplikacemi a systémy, které potencionální uživatelé používají pro provoz své infrastruktury a podnikových procesů (Operation support system).

V oblasti NFV MANO probíhá v současnosti rozsáhlý vývoj a existuje několik projektů, které se tím zabývají. V článku [23] je nabídnut zajímavý přehled. V kapitole 3.4 je uveden přehled možných přístupů k managementu a konfiguraci jednotlivých VNF.

3.2. Dostupné prostředky pro NFV Infrastrukturu

Pro tvorbu NFV infrastruktury bude v této práci využívána cloudová platforma. V přehledu [1], který byl zmíněn v úvodu, je také uvedeno, že mezi nejpoužívanější řešení pro cloud patří OpenStack a řešení od společnosti VMware. Tyto dvě jsou dále popsána podrobněji.

3.2.1. OpenStack

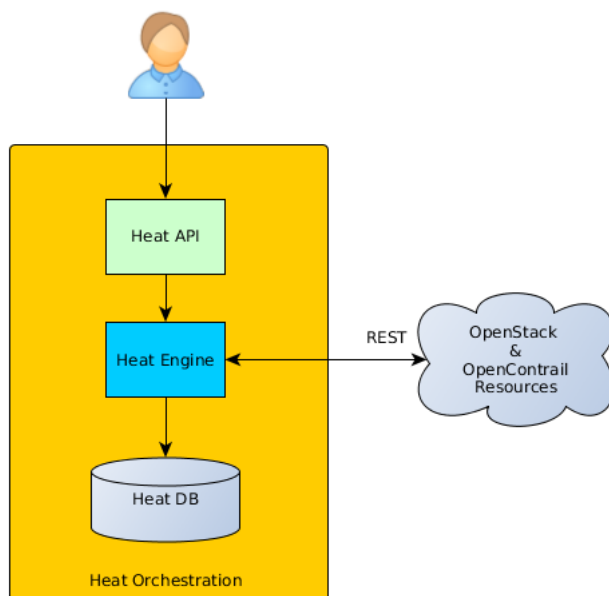
OpenStack [24] je open-source platformou umožňující postavit cloud, který může být nainstalován na běžném hardwaru. Toto řešení má za cíl vytvořit dostupnou cloudovou platformu, která bude splňovat všechny potřeby privátních a veřejných cloudů nezávisle na velikosti řešení.

Celá stavba systému OpenStack se skládá z několika na sobě nezávislých projektů (modulů), které řeší různé oblasti cloudové platformy. Tyto projekty mezi sebou komunikují pomocí otevřených API a mohou být spravovány pomocí dashboardu. Celé administrace OpenStacku může být prováděna přes webové rozhraní, příkazovou řádku či přímo pomocí příkazů zaslaných do API. Celé toto řešení se vyznačuje jednoduchostí implementace, škálovatelností a rychlým vývojem nových vylepšení. Hlavními moduly OpenStacku jsou:

- Keystone - identifikační služba používaná OpenStackem pro autorizaci a autentizaci. Ověřování probíhá pomocí tokenů. Uživatel přihlášením odesílá žádost na Keystone, který tento modul zpracuje, zjistí pověření a vytvoří token. Vytvořený token je poté odesílán s žádostí do ostatních služeb. Zde dojde ke komparaci tokenu se současnou přístupovou politikou a dojde ke zjištění, zdali má uživatel dostatečná oprávnění pro provedení požadovaného úkonu.
- Glance - služba umožňující práci s virtuálními diskovými obrazy (imagy). Tyto obrazy mohou být uloženy na mnoha různých místech od lokálních systémových disků až po distribuované souborové systémy, jako je OpenStack Storage.
- Nova - tento modul poskytuje výpočetní služby. Umožňuje tedy běh několika instancí virtuálních strojů na několika hostitelských strojích, na kterých je nainstalována služba OpenStack compute. OpenStack podporuje hypervizory KVM, QEMU, VMware ESX, Hyper-V, Xen.
- Neutron - je služba pro správu všech síťových aspektů OpenStacku. Jedná se tedy o SDN komponentu. Neutron podporuje možnost rozšíření o tzv. pluginy, které umožňují využívat řešení třetích stran pro síťování.
- Cinder - poskytuje infrastrukturu pro mapování volumů v OpenStacku.
- Heat - umožňuje automatizovanou orchestraci virtuálních strojů na základě vytvořených šablon.
- Horizon - představuje dashboard, který umožňuje cloudovým administrátorům a uživatelům spravovat různé zdroje a služby OpenStacku. Dashboard umožňuje interakci s OpenStackovým kontrolerem prostřednictvím API.

Další informace lze nalézt například v [25]. V této práci jsou vzhledem k NFV infrastruktuře nejzajímavějšími Neutron a Heat moduly.

Heat je projekt, který je určen pro Orchestraci. Má za úkol automatické vytvoření požadovaných resourců (instancí, sítí, atd) podle předdefinovaných scénářů tzv. heat templatů. Obrázek č. 3.5 znázorňuje jeho funkci. [26]



Obrázek 3.5.: Popis heat orchestrace

Dalším projektem, který v rámci NFV a OpenStacku důležitý je Neutron, který slouží pro práci se sítěmi. Neutron má podporu pro rozšiřující pluginy, které mohou využít SND řešení. Funkce OpenStacku se značně mění dle použitého pluginu, kterých je více než 20. Pokud se podíváme na [27], tak mezi nejpoužívanější patří tzv. Vanilla Neutron a OpenContrail.

3.2.1.1. Vanilla Neutron

Vanilla Neutron nabízí základní funkcionalitu pro networking v prostředí OpenStack, kterou uživatel může vyžadovat. Je to z důvodů toho, že OpenStack původně vyšel z AWS (Amazon Web Services) a jeho cílem bylo sjednotit privátní a veřejný cloud. Dnes do Neutronu přibývají další funkce jako je například VPNaaS.

Problém samotného Neutronu je, že prozatím nemá podporu pro NFV. Proto byla vytvořena iniciativa OP-NFV [28], která se snaží vytvořit otevřený framework pro NFV založený na OpenStacku. Přestože tento projekt má velký potenciál, tak

v době psaní této práce je v začátcích a není vhodný pro produkční nasazení.

3.2.1.2. OpenContrail

OpenContrail je jeden z nejpoužívanějších komerčně dostupných řešení pro networking pro OpenStack. Obsahuje všechny potřebné komponenty pro virtualizaci sítí v cloudovém prostředí - SND controller, virtuální router, analytický engine a REST API. [29]

OpenContrail není pouze SDN řešení, ale je to i řešení pro NFV. OpenContrail obsahuje funkci Service Chainingu. Tím pádem umožňuje dynamicky vytváření instancí, které mohou sloužit jako VNF. Zároveň také dokáže řídit datový tok z ostatních instancí resp. virtuálních sítí k nimž je VNF připojena tak, aby procházela právě danou VNF. V poslední verzi dále přináší nové funkce jako BGPaaS a Physical Service Chaining. Je to tedy komplexní řešení vhodné pro produkční nasazování. Více informací o OpenContrailu lze nalézt v dokumentaci [30].

3.2.2. VMware vCloud Suite

Společnost VMware [31] se původně zabývala vývojem předního virtualizačního nástroje. Postupně nabrala do svého portfolia i další služby a rozšířila působnost obecně na virtualizaci a služby s ní úzce spojené. Mezi ně dnes patří i cloudová platforma.

VMware vCloud Suite je právě určený pro vytváření privátních cloudů. Je to řešení poskládané z jednotlivých produktů společnosti VMware. Jeho hlavními komponentami jsou:

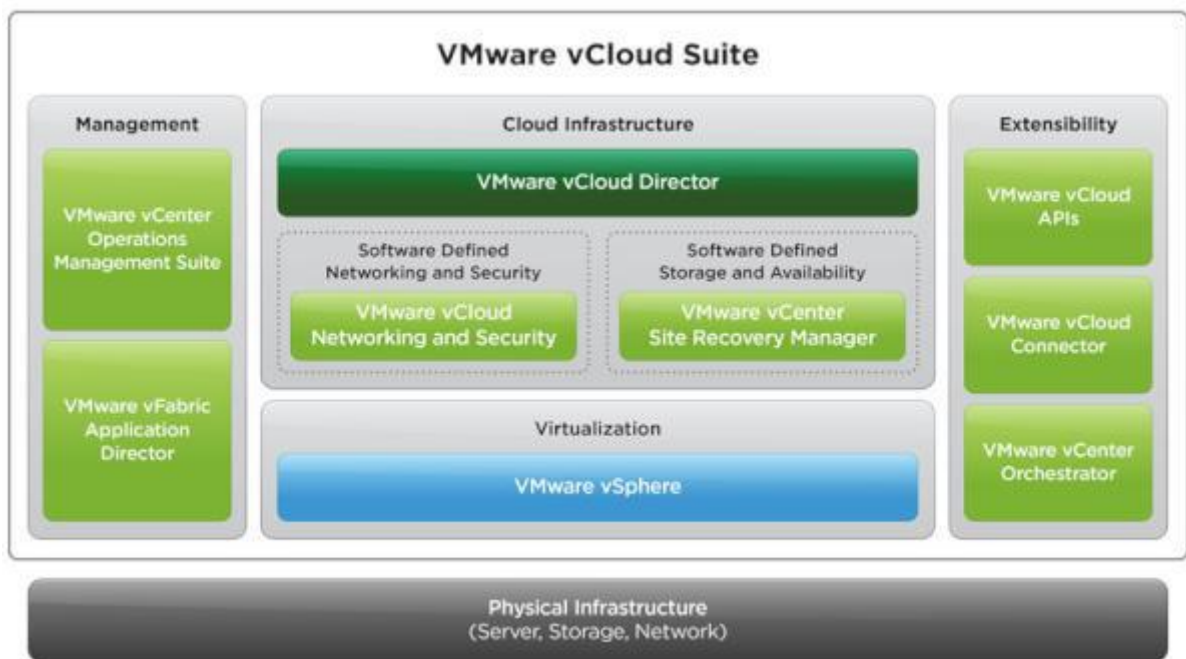
- VMware vCloud Director - je jedna ze základních součástí potřebných pro vytvoření privátního cloudu ve stylu VMwaru. Umožňuje vytvořit a doručovat koncovým zákazníkům infrastrukturu jako službu. Je propojen a přímo spolupracuje s VMware vSphere center.
- VMware vSphere - tento produkt slouží pro vytvoření virtualizované infrastruktury. Je to sdružení více komponent. Ty nejhlavnější jsou:

- VMware ESX (ESXi) - je to bare-metal hypervisor.
- VMware vCenter Server - umožňuje efektivní a pokročilejší správu virtualizovaného prostředí, bez ohledu na jeho velikost. Jedná se např. o snadné vytváření nových virtuálních počítačů, jejich klonování nebo importování z jiného úložiště.
- VMware vSphere Client - je určený pro dálkovou správu hostitelů ESXi. Připojit se můžeme prostřednictvím vCenter serveru nebo přímo přes ESXi server.
- VMware vCloud Networking and Security - poskytuje síťování a bezpečnost pro virtuální prostředí. Poskytuje mnoho síťových funkcí a poskytuje framework pro integraci řešení třetích stran.
- VMware vShield - představuje možnost zabezpečení v prostředí VMware vSphere. Může být konfigurován pomocí vShield Managera, který umožňuje centrální správu přes webové rozhraní, vSphere klient plug-in, nebo command line interface (CLI).
- VMware vCenter Chargeback - slouží pro monitorování virtuálních strojů, které následně může být účtováno.

VMware VCloud Suite má hierarchickou strukturu, kterou znázorňuje obrázek č. 3.6. Více informací o tomto řešení či jeho částech nabízí např. [32] .

Celé vmware řešení je proprietární. To znamená, že kód pro jednotlivé komponenty je uzavřený a není do něj přístup. S tím samozřejmě souvisí i licence. Pro používání jakéhokoli produktu od společnosti VMware v produkčním prostředí je potřeba zakoupení licence. Všechny produkty této společnosti se vyznačují tím, že je vše ovládané z grafického uživatelského prostředí bez nutnosti použití příkazové řádky. Je však toto řešení dostatečně standardní pro NFV Infrastrukturu?

Pokud se podíváme na možnosti orchestrace, tak je zde oproti předchozímu řešení zvolený odlišný přístup. Přestože VMware poskytuje REST API, tak vše je orientované na ovládání zkrze dashboard. To znamená, že automatizace či před-



Obrázek 3.6.: Schéma VMware vCloud Suite, převzato z [33]

definování scénáře pro vytváření resourců zde spočívá v naklikání celého workflow v přehledném GUI.

Pro networking v této cloudové platformě existuje tzv. VMware NSX. Je to SDN Controller, které je opět velmi často používané i s cloudovou platformou OpenStack. Přestože opět obsahuje vše potřebné pro vytváření a správu virtuálních sítí, tak zde není přímá podpora NFV, která by usnadnila práci s jednotlivými VNF.

3.3. Dostupné možnosti pro VNF

Protože v této práci mají být ukázány příklady pro vytváření a management VNF. Je nutné najít software, který může být použit jako VNF. Protože existuje celá řada síťových funkcí a bylo by nad rámec této práce zmínit všechna jejich řešení, tak bylo nutné výběr zúžit výběr na dvě nejčastěji používané síťové funkce. Těmi jsou firewall a load balancer. Dále je tedy uveden software, který je možné použít pro Firewall as a Service (FwaaS) a Load balancer as a Service (LbaaS).

3.3.1. Firewall as a Service

Pokud se podíváme na vendory ve zdroji [34], jejichž firewally jsou nejvíce využívány, tak zjistíme, že mnozí již chápou důležitost NFV a své produkty začínají poskytovat jako virtuální instance, které se dají použít pro účely VNF v této práci. Jedná se například o tyto:

- Fortigate-VM - Fortigate Virtual Appliances je řešení pro cloudové prostředí od společnosti Fortinet. Nabízí stejně funkce pro firewall jako jsou obsaženy ve Fortigate fyzických zařízeních. [35]
- Juniper vSRX - Jde o firewall od společnosti Juniper, který je obdobou jejich fyzického zařízení Juniper SRX. Jde virtuální instanci poskytující funkce pro firewall, routing a pokročilé bezpečnostní funkce pro poskytovatele telekomunikačních služeb a větší společnosti. Toto VM je určené pro privátní, public i hybrid cloud. [36]
- Cisco virtual ASA - Společnost Cisco nabízí Adaptive Security Virtual Appliance (ASAv), která obsahuje stejný software jako fyzické ASA zařízení a většinu funkcí pro firewall, routing a VPN. [37]
- PFSense - PFSense je open-source projekt, který má za cíl poskytnout firewall postavený na operačním systému FreeBSD, který může běžet na klasické architektuře jednodeskových počítačů. Toto řešení poskytuje všechny důležité vlastnosti komerčních firewallů, má jednoduché ovládání a je to otevřené řešení. [38]

3.3.2. Load balancer as a Service

Na trhu s virtuálními load balancery je situace podobná. Zde vendori také začínají poskytovat virtuální instance či software. Pro správné fungování load balancu je však nejlepší, pokud existuje integrace či plugin pro danou cloudovou platformu či používané SDN. Výše zmíněný VMware i OpenStack podporují následující software sloužící jako LbaaS.

- HAproxy – Je velmi rychlé a spolehlivé řešení nabízející vysokou dostupnost, load balancing a proxy pro aplikace založené na TCP a HTTP. Jedná se de-facto o standartní opensource load balancer, který občas bývá součástí některých linuxových distribucí. Velice oblíbené je jeho využití práce cloudových platformách, kde je jako defaultní možnost pro load balancing. Více informací poskytuje [39] .
- AVI networks - Je celkem nová společnost poskytující platformu, která poskytuje automatizované aplikační služby zahrnující load balancing, aplikační analýzu a prediktivní auto škálování. Platforma je postavena na softwarově definovaném principu a tím pádem může být nasazena klasickou x86 platformu serverů. Více informací obsahuje [40] .

3.4. Možnosti prostředky pro Management a Orchestraci VNF

Poslední částí referenční architektury NFV je management a orchestrace VNF. Přestože se jedná o nejkomplexnější část architektury, tak v této práci se situace o něco zjednodušila použitím cloudové platformy. Díky ní totiž není nutné hledat řešení pro úlohu, kterou plní Virtualized infrastructure manager, neboť pokud bude provedena správná volba této platformy, tak v ní bude již obsažena. Zároveň je třeba říci, že zde není tedy nutné se příliš starat o to, kde budou jednotlivá VNF vytvořena, přestože z hlediska výkonu to může hrát roli, jak je například popsáno v [41] .

Dále je nutné zmínit, že není nutné realizovat roli NFV orchestratoru, protože by to bylo nad rámec cílů této práce. Jediným problémem, pro který je tedy nutné navrhnout řešení je management jednotlivých síťových funkcí. Roli, kterou zastává VNF manager. Tou je tedy management jednotlivých VNF a softwaru, který na nich poběží.

Management počítačové sítě je sama osobě komplexní činnost skládající dle [42] se ze:

- Správa poruch a chyb (ang. Fault management)
- Správa konfigurace (ang. Configuration management)
- Účetní a evidenční správa (ang. Accounting management)
- Správa výkonu (ang. Performance management)
- Správa bezpečnosti (ang. Security management)

Přestože všechny tyto části jsou důležité, tak kromě správy konfigurace jsou to detekční a monitorovací úkony, které mohou být do určité míry i součástí implementace cloudové platformy. Avšak správa konfigurace je hlavní část, pro kterou je nutné nalést řešení v této práci, protože většina softwaru pro VNF uvedené v předchozí části bude mít jiné možnosti pro jejich správu konfigurace. Existuje zde tedy několik přístupů, jak k ní přistupovat. Některé jsou zmíněné v předchozí práci [43]. Pro účely této práce lze uvažovat o následujících:

- Předdefinovaný image či aplikace - V tomto případě je veškerá potřebná konfigurace součástí dodávaného image. Pokud je požadována jednoduchá síťová funkce, jejíž konfigurace se v čase nebude měnit, pak je toto nejlepší způsob.
- Využití scriptování / API - Pokud daná VNF poskytuje API či podporuje skriptovací rozhraní, pak je možné ho využít a dopravit, tak požadovanou konfiguraci.
- Standardizované protokoly - Dnes již také existují relativně standardizované protokoly určené právě pro správu konfigurace. Jedním je například protokol NETCONF, jak je uvedeno v [42]
- Konfigurační management - Další možností je využití konfiguračního managementu, který se dnes velmi často využívá při správě velkého množství serverů. Zde je však nutné, aby daná VNF podporovala danou metodu správy, kterou využívají jednotlivé konfigurační managementy. Může být

například vyžadováno, aby daná VNF obsahovala agenta, který bude pro konfiguraci využíván. Dnes existuje několik řešení pro konfigurační management jako je např. SaltStack, Puppet či Ansible.

4. Požadavky a návrh prostředí pro NFV a VFN

V předešlé kapitole byla vysvětlena základní problematika, která souvisí s virtualizací síťových funkcí, cloud computingem a softwarově definovanými sítěmi. Zároveň byla popsána referenční architektura frameworku pro virtualizaci síťových funkcí a popsána existující řešení vhodná pro jednotlivé části architektury.

Tato kapitola bude již věnována konkrétnímu návrhu platformy pro testování NFV a VNF. Nejprve jsou zde popsány jednotlivé požadavky související s návrhem platformy pro NFV a následně jsou popsány požadavky i na jednotlivé VNF. Na základě těchto požadavků je poté navrhuta a popsána výsledná architektura.

4.1. Požadavky a výběr platformy pro NFV Infrastrukturu

V předchozí kapitole bylo řečeno, že existuje několik možností, jakými může být NFV infrastruktura vytvořena. Z tohoto důvodu musí být definované specifické požadavky, které by měla splňovat NFV infrastruktura pro účely této práce. Podle nich následně může být zvolena specifická technologie. Na základě již uvedených informací o NFV a cílů této práce byli identifikovány následující požadavky:

- Otevřenost řešení - Vzhledem k tomu, že NFV je založeno na myšlence přechodu z proprietárních hardwarových boxů k standardním softwarově defino-

vaným funkcím, tak je rozumné využít opensource technologie i pro NFV Infrastrukturu.

- Možnost automatizace a vytváření šablon - Jedním z cílů práce je VNF řešení, které by mohlo sloužit jako blueprint pro ostatní uživatele. Proto musí být zvolené technologie, které mají podporu vytváření šablon. Těmi může být následně automaticky vytvořeno celé či alespoň část řešení pro VNF. Zároveň však jejich tvorba musí být dostatečně flexibilní a jednoduchá, aby ji zvládli i uživatelé.
- Podpora SDN - Dalším důležitým požadavkem je podpora softwarově definovaných sítí. Jak již bylo řečeno, tak přestože NFV a SDN jsou odlišné technologie, tak se velice dobře doplňují. Z tohoto důvodu by zvolená platforma pro NFV Infrastrukturu měla podporovat i SDN. Díky tomu následně může být využíván service chainig pro jednotlivá VNF.

Na základě těchto požadavků je v následující tabulce uvedeno srovnání cloudových platform z předchozí kapitoly.

	VMware vCloud Suite	OpenStack
Typ	opensource	proprietary/licence
Podpora SDN	ANO (VMware NSX)	ANO (OpenContrail)
Service Chaning	ANO	ANO
Tvorba šablon pro orchestraci	Tvorba v GUI	Tvorba v YAML formátu

Tabulka 4.1.: Srovnání VMware vCloud Suite a OpenStacku na požadovaných parametrech

Na základě těchto informací bylo rozhodnuto, že pro NFV infrastrukturu bude využita cloudová platforma OpenStack.

4.2. Požadavky a výběr řešení pro VNF

Dále je potřeba rozhodnout, který software bude použit pro implementaci jednotlivých VNF. Proto byly opět definované požadavky, které by potenciální software měl splňovat. Tyto požadavky jsou:

- Dostupnost - Pro to, aby bylo vůbec možné uvažované řešení použít, tak je nutné, aby bylo dostupné pro testování. Dostupnost může být ve formě opensource či určité formy trial licence.
- Kompatibilita s platformou x86 - Dalším požadavkem musí být kompatibilita s platformou x86. Je tedy nutné, aby mohlo být využito klasických serverů pro řešení.
- Úspěšné spuštění na virtualizační platformě - Pro účely této práce musí být ověřeno, že dané řešení je schopné běžet na daném hypervizoru, který bude použit s OpenStackem. Tedy s hypervizorem KVM.
- Integrace s OpenStackem - V případě LbaaS je požadované, aby vybraný software měl určitou formu integrace s prostředím OpenStacku. Je to z důvodu lepší automatizace VNF.

Následující tabulka nabízí přehled toho, zda daný software splňuje určené požadavky.

LbaaS	HAproxy	AVI networks
Dostupnost	ANO - opensource	ANO - Trial licence
Integrace s OpenStackem	ANO (default)	ANO - plugin
Úspěšné spuštění	ANO	ANO
Kompatibilita x86	ANO	ANO

Tabulka 4.2.: Přehled softwaru pro LbaaS

Z výše uvedených informací vyplývá, že pro implementaci LbaaS bude použita HAproxy a platforma, kterou poskytuje firma AVI networks. V případě softwaru

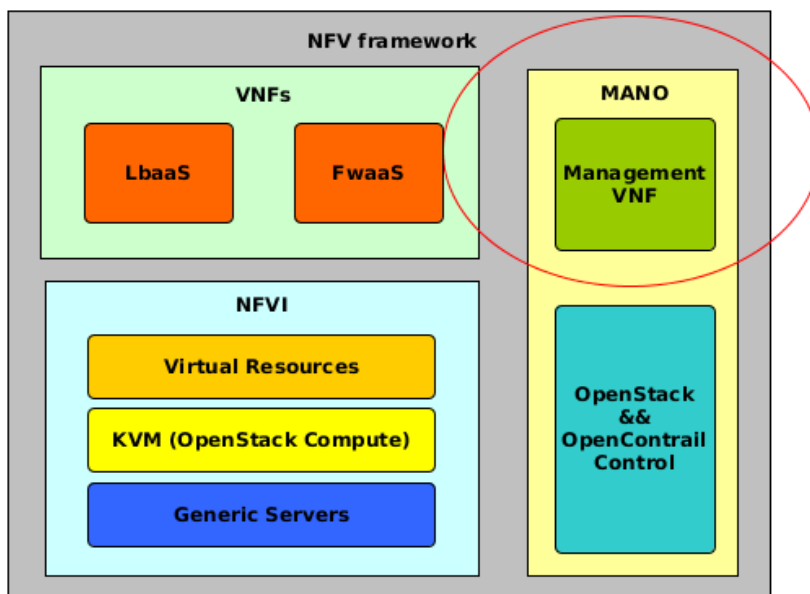
FwaaS	Fortigate VM	Juniper vSRX	Cisco ASAv	PfSense
Dostupnost	ANO - Trial	ANO - Trial	NE - Licence	ANO - opensource
Úspěšné spuštění	ANO	NE	NE	ANO
Kompatibilita x86	ANO	ANO	ANO	ANO

Tabulka 4.3.: Přehled softwaru pro FwaaS

pro FwaaS je zřejmé, že bude využit PfSense a Fortigate VM v 15 denní trial verzi. Cisco ASAv nebylo možné testovat kvůli licenčním poplatkům a Juniper vSRX se bohužel nepodařilo zprovoznit v KVM hypervizoru na testovací infrastruktuře.

4.3. Výsledná architektura použitého frameworku

V předchozích částech byly vybrány technologie, které jsou použity v návrhu pro NFV framework. Na obrázku č. 4.1 je znázornění celého framework s vyobrazením všech použitých částí.

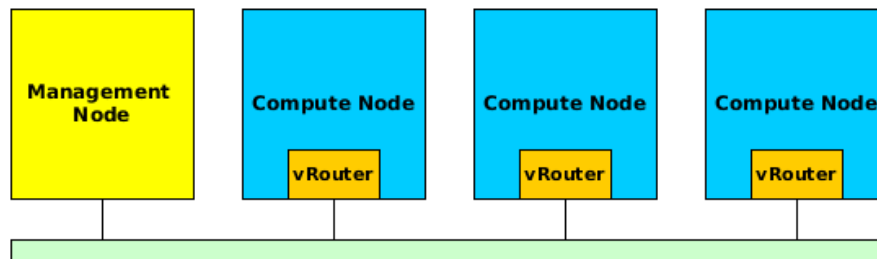


Obrázek 4.1.: Výsledná architektura NFV

Z obrázku je patrné, že celou NFV Infrastrukturu tvoří OpenStack s SDN řeše-

ním OpenContrail. Ty zasahují i managementu a orchestrace na nejnižší úrovni. Jako hypervizor byl použit KVM. Jako VNF služby budou poskytovány Load Balancer a Firewall. Poslední částí, kterou se bude zabývat zbytek práce je management jednotlivých VNF.

Pro fyzické vytvoření frameworku bylo využito 4 serverů. Jeden slouží jako Control node, na kterém jsou řídicí služby OpenStacku a OpenContrailu společně s jejich dashboardy. Zbylé 3 servery jsou použity jako Compute nody. Na nich nainstalovaný hypervizor společně s compute rolemi OpenStacku. Dále je na nich i OpenContrail vRouter. Pomocí něho je vytvořený tzv. Overlay network, který používají vytvořené instance.



Obrázek 4.2.: Schéma výsledné fyzické topologie pro NFV

5. Testovací scénáře a realizace pro VNF a NFV

V předchozí kapitole byla popsána oblast virtualizace síťových funkcí a její architektura. Také byly popsány jednotlivé technologie, které budou v této kapitole použity pro vytvoření NFV frameworku. Ten bude následně využit k realizaci ukázkových VNF. Jak již bylo řečeno, tak pro každou VNF zde bude uveden příklad jejího použití a jakým způsobem je možné vytvořit automatizované řešení pro VNF nad navrženým NFV frameworkem.

5.1. Scénáře pro použití vybraných VNF

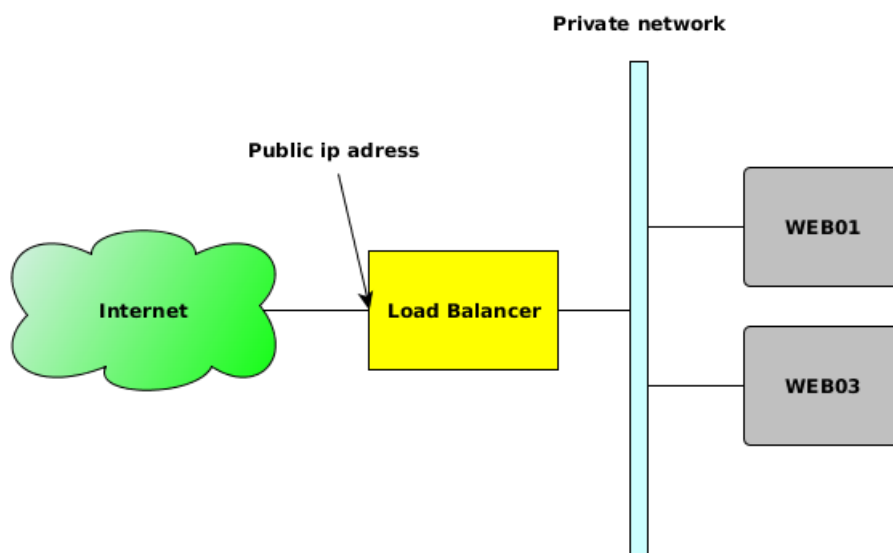
Před samotnou implementací je nutné vysvětlit, jakým způsobem budou VNF využívat uživatelé. Toto by mělo sloužit pro lepší pochopení daných příkladů a usnadnit realizaci jednotlivých VNF. Dále v této sekci budou postupně uvedeny jednotlivé scénáře pro všechna uvedené VNF v této práci.

5.1.1. Scénář LbaaS

Jedním z často využívaných síťových funkcí je load balancing. Pokud chce uživatel v cloudu provozovat nějaký druh webové služby, která musí být vysoce dostupná nebo bude velice vytížená, tak bude ve většině případů potřebovat využít více než jeden server. Pro rozdělení zátěže mezi tyto servery je obvykle nutné využít load balancer. Ten bude spravovat příchozí komunikaci a distribuovat ji mezi několik serverů. Tím bude zajištěna rozloha zátěže a zajištěn bezvýpadkový pro-

VOZ.

Uvažujme tedy jednoduchý příklad, ve kterém bude uživatel požadovat použití dvou serverů, na kterých poběží webová služba v HA. Pro tuto službu bude chtít využít cloudovou platformu s podporou LbaaS. Uživatel samozřejmě požaduje, aby se celá infrastruktura a nastavení webové aplikace proběhlo automaticky s co možná nejméně manuálními zásahy. Celý proces by se tedy v praxi měl skládat z výběru vhodného templatu, zadáním požadovaných vstupních parametrů (IP adresy, názvy instancí, atd.) a následného spuštění templatu, který vše vytvoří. Obrázek č. 5.1 zobrazuje požadovaný výstup takového procesu.



Obrázek 5.1.: Firewall as a Service

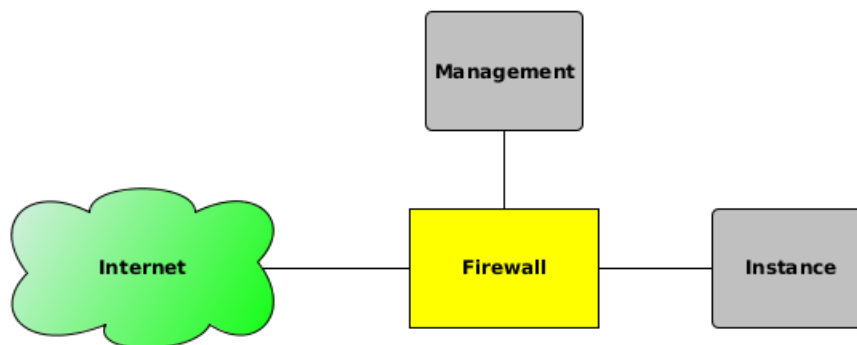
Je tedy vidět, že VNF pro load balancing by měla být již integrovanou součástí návrhu cloudové aplikace, resp. návrhu cloudového templatu pro orchestraci. V případě HAproxy by toto neměl být problém, protože je to standartní součást OpenStacku a OpenContrailu. Mělo by tedy jít využít standartního heat templatu, ve kterém bude nadefinované vše potřebné pro Load Balancer. Otázkou je, zda bude možné využít stejný postup i případě AVI networks řešení.

5.1.2. Scénář FwaaS

Další z často využívaných síťových funkcí je firewall, resp. síťové funkce, které dokáží řídit a zabezpečovat síťový provoz mezi různými sítěmi. Zjednodušeně se dá říct, že slouží jako kontrolní bod, který definuje pravidla pro komunikaci mezi sítěmi, které od sebe odděluje.

Příkladů užití FwaaS v cloudovém prostředí existuje značné množství, protože uživatelé mohou mít různé požadavky na řízení datového provozu. Uvažujme však nejjednodušší příklad, kterým je překlad adres tzv. NAT. V tomto případě uživatel chce vytvořit virtuální instanci, která by měla mít konektivitu k externí síti, ale zároveň nechce, aby daná instance byla dostupná z této externí sítě.

Otázkou zde je, jakým způsobem by měl být daný firewall nakonfigurován? Jak bylo popsáno kapitole 3.4, tak existuje více možností. Konfigurace by mohla být doručena jako součást orchestračního templatu, např. v podobě skriptu, který by se provedl po spuštění firewall instance. Tento přístup však nemusí existovat podpora ze strany PfSense a Fortigatu. V tom případě bude muset být využita jiná metoda, pro kterou však ve většině případů musí být vyhrazený dedikovaný interface, určený právě pro management. Tento interface může být využit i pro manuální zásahy do konfigurace. Obrázek č. 5.2 zobrazuje schématické zapojení pro FwaaS.



Obrázek 5.2.: Firewall as a Service

5.2. Realizace VNF pro LbaaS

Zde již budou vysvětleny jednotlivé kroky související s realizací VNF pro LbaaS. Nejprve je ukázáno řešení s využitím HAproxy. Popsaný je zde obecný popis funkce haproxy, resp. LbaaS Neutronu a popis toho, jak byl využit pro tvorbu VNF. Po této ukázce následuje popis a vysvětlení realizace VNF související s platformou od AVI networks.

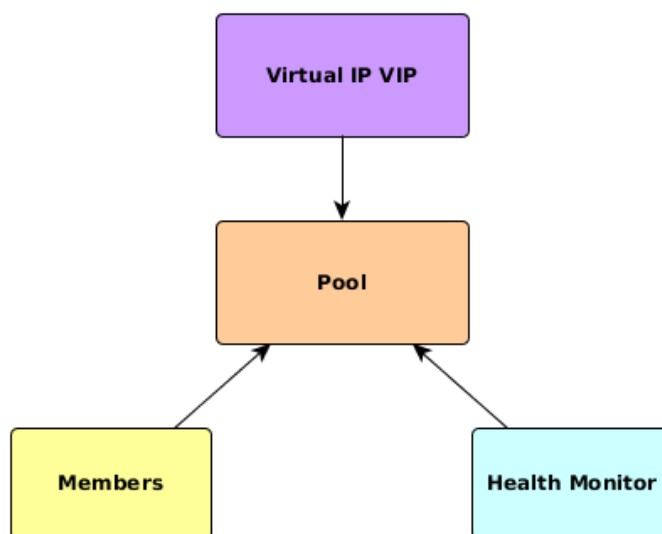
5.2.1. HAproxy - Neutron HAproxy agent

OpenStack Neutron ve své implementaci obsahuje službu LBaaS. Je to jedna z jeho pokročilou služeb, která umožňuje použít jeden soubor API k ovládání load balanceru od poskytovatelů třetích stran. Jedinou podmínkou je, aby toto API implementovali. Toto velice zjednodušuje uživatelům OpenStacku ovládání load balancerů a odpadá díky tomu nutnost seznamování se s implementací a konfigurací těchto různých řešení, která mohou být velmi specifická a odlišná. HAproxy je defaultní backend sloužící pro LBaaS pro OpenStack s využitím OpenContrailu jako SDN řešení.

Load balancer se v Neutron LBaaS skládá ze základních 4 objektů, které jsou vzájemně provázány. Tyto objekty jsou:

- Pool - Obsahuje základní parametry pro load balancer. Jako je např. síťový rozsah, ve kterém budou webové servery či metodu pro distribuci zátěže (Round Robin, Least connection, atd.)
- Virtuální IP (VIP) - ip adresa, na kterou přichází komunikace
- Member - Je konkrétní virtuální instance, která je členem poolu např. webový server. Označení membra je pomocí jeho ip adresy.
- Monitor - Periodicky kontroluje stav jednotlivých serverů a aplikací. Kontrola může být pomocí pingu či http a https dostupnosti.

Obrázek č. 5.3 zachycuje jednotlivé závislosti mezi těmito objekty.



Obrázek 5.3.: Neutron LbaaS

Celý proces probíhá tak, že každý virtuální server, který je asociovaný s daným poolem z něj obdrží IP adresu. Když přijde na VIP nějaký dotaz na danou webovou aplikaci, tak je tento dotaz předán dál na jednu z těchto přiřazeným IP adres. Pokud nastane s aplikací či serverem nějaký problém, který zachytí monitor, tak load balancer ip adresu tohoto serveru přestane posílat komunikaci, dokud není vše zase v pořádku.

Z výše uvedených informací vyplývá, že je nutné správně nadefinovat jednotlivé komponenty v heat templatě tak, abychom dosáhli požadovaného chování. Tím bude zajištěné správné a automatické řešení pro VNF.

5.2.1.1. Heat template pro haproxy

Celý heat template pro LbaaS s HAproxy v sobě obsahuje několik prostředků, které se po jeho spuštění pokusí heat engine vytvořit. Tvorba heat templaty vychází z informací uvedených v dokumentaci [26]. Protože celý heat template je značně dlouhý, tak zde nebude ukazován celý jeho kód, ale pouze části týkající se load balanceru. V případě zájmu lze celý template nalést v příloze. Template se tedy skládá z:

- Privátní síť - K této síti jsou připojeny obě webové instance, load balancer a

router. Součástí je definice toho zdroje jsou i subnet, který má dále parametry týkající se DHCP ip adres.

- 2 x web instance - jedná se o virtuální instance s operačním systémem Ubuntu 14.04. Po spuštění heat templatu se na tyto instance nainstaluje Apache server a vytvoří se index.html. Díky tomu je možné následně otestovat zda load balancer distubuje komunikaci mezi těmito dvěma servery.
- router - toto je Neutron router implementující SNAT. V tomto příkladě je využíváný webovými servery pro konektivitu k Internetu. Toto je nutné pro nainstalování programu Apache na webové servery.
- public síť - toto je veřejná síť, ze které je získána VIP pro load balancer. Na tuto VIP bude dále asociována floating ip.
- members - po vytvoření instancí je nutné jejich přidání do poolu jako members. Pokud webová aplikace na serverch využívá jiný port než port 80, je možné ho zde změnit.

```
lb_pool_member_instance_01:
  type: OS::Neutron::PoolMember
  properties:
    address: { get_attr: [ instance_01 , first_address ] }
    admin_state_up: True
    pool_id: { get_resource: lb_pool }
    protocol_port: 80
    weight: 1
```

Ukázka kódu 5.1: Healt monitor

- health monitoring - zdroj pro monitoring. Dle zvolených parametrů je vidět, že každých 5 sekund bude poslán ping na servery a bude se čekat 5 sekund na odpověď. Pokud nepříjde, tak load balancer usoudí, že je daný server není v pořádku a přestane na něj přeposílat komunikaci.

```
lb_ping_healt_monitor:
  type: OS::Neutron::HealthMonitor
  properties:
    admin_state_up: True
    delay: 5
    max_retries: 1
    timeout: 5
    type: PING
```

Ukázka kódu 5.2: Healt monitor

- pool - jedná se o definování poolu pro load balancer. Na ukázce je vidět, že byla zvolena metoda Round Robin. Tato metoda byla zvolena kvůli co nejjednoduššímu testování tohoto templatu.

```
lb_pool:
  type: OS::Neutron::Pool
  properties:
    admin_state_up: True
    lb_method: ROUND_ROBIN
    name: { get_param: lb_name }
    protocol: HTTP
    monitors:
      - { get_resource: lb_ping_healt_monitor }
    subnet_id: { get_resource: private_subnet }
  vip:
    protocol_port: 80
    admin_state_up: True
    subnet: { get_resource: public_subnet }
```

Ukázka kódu 5.3: Healt monitor

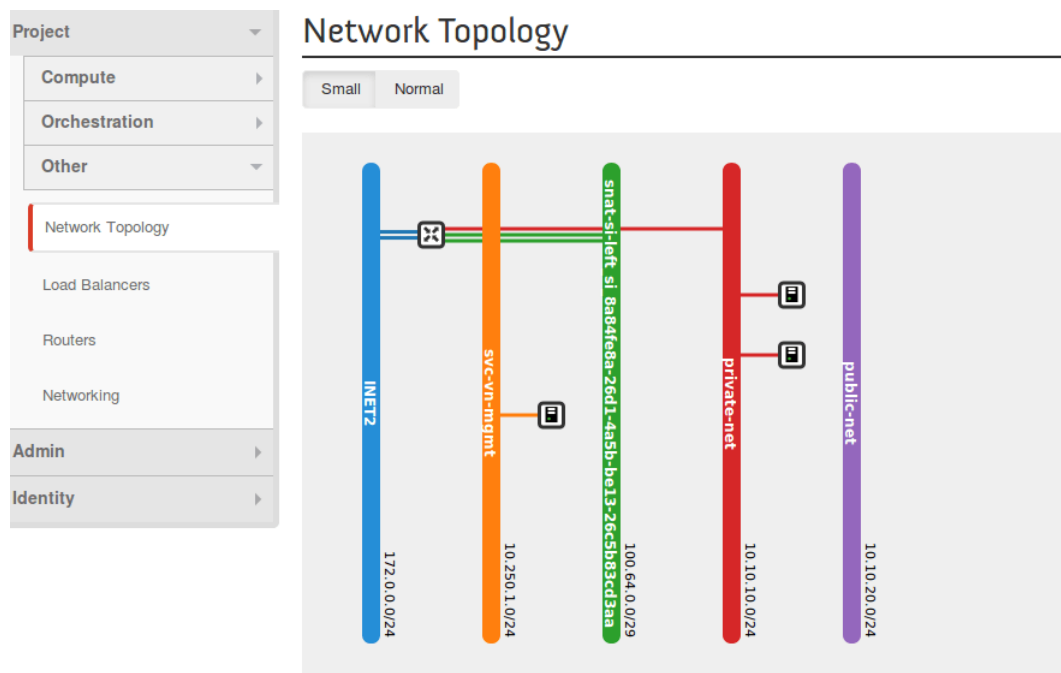
Takto byl vytvořen heat template, který vytvoří LbaaS pomocí HAproxy. Nyní je nutné tento template otestovat, zda funguje správně.

5.2.1.2. Testování heat templatu pro haproxy

V reálném případě by si uživatel heat template vybral z katalogu. Avšak v rámci testování bude heat template spouštěn pomocí příkazu v terminálu:

```
heat stack-create -f heat/templates/lbaas_template.hot -e
  heat/env/lbaas_env.env lbaas
```

Tento příkaz vytvoří všechny prostředky uvedené v templatu pro load balancing. Na obrázku č. 5.4 je zobrazen screenshot vytvořené topologie z OpenStack dashboardu. Jsou zde vidět vytvořené servery a sítě. Není zde zobrazen load balancer, protože tato vizualizace tento prvek nezobrazuje. Lze ho nalézt v jiné části dashboardu, ale pro názornost bude rovnou otestováno jeho správné chování.

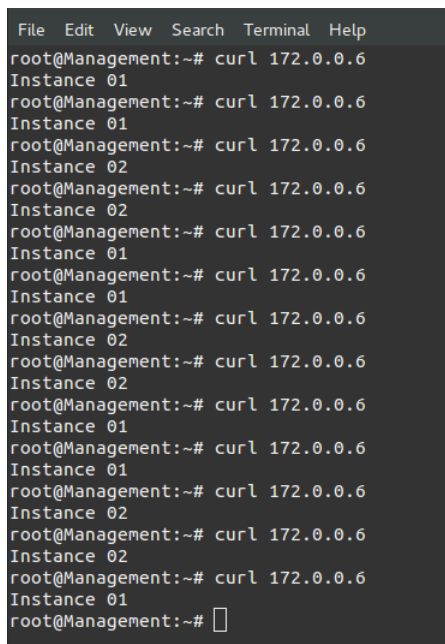


Obrázek 5.4.: Vytvořená síťová topologie

Otestování správného chování virtuálního load balanceru, lze provést opakovaným dotazem na právě vytvořené webové servery. Tím bude zároveň otestována jejich správná konfigurace. Pokud by totiž nevrátili správnou odpověď je možné, že chyba může být i zde.

Dotaz na webové servery byl proveden pomocí příkazu curl, kterému byla dána

jako parametr public adresa load balanceru. Celý výstup toho testování znázorňuje obrázek č. 5.5.



```
File Edit View Search Terminal Help
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~# curl 172.0.0.6
Instance 02
root@Management:~# curl 172.0.0.6
Instance 02
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~# curl 172.0.0.6
Instance 02
root@Management:~# curl 172.0.0.6
Instance 02
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~# curl 172.0.0.6
Instance 02
root@Management:~# curl 172.0.0.6
Instance 02
root@Management:~# curl 172.0.0.6
Instance 01
root@Management:~#
```

Obrázek 5.5.: Test konektivity a load balancingu

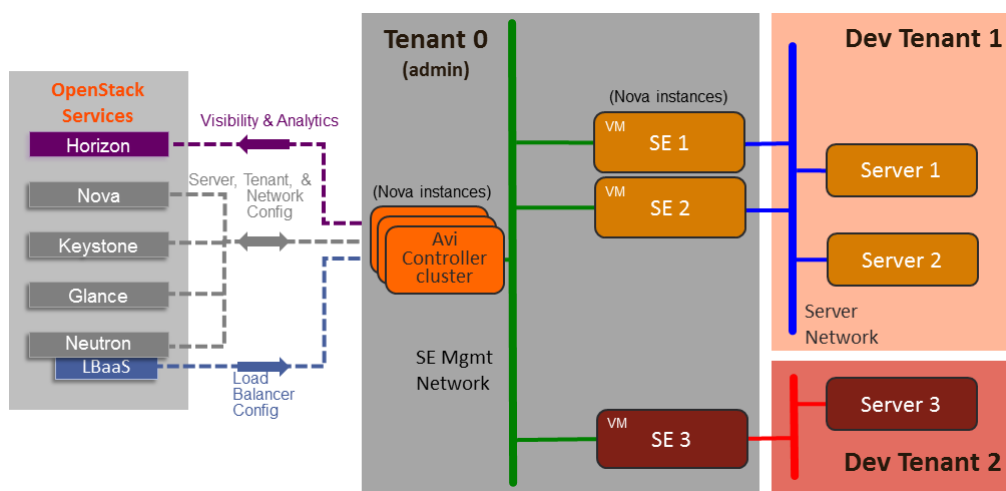
Po několika takovýchto dotazech na webové servery je vidět, že odpověď přichází střídavě od obou webových serverů. Probíhá mezi nimi tedy load balancing metodou round robin tak, jak bylo požadováno.

5.2.2. AVI networks

Druhým řešením pro LbaaS, které bude popsáno a otestováno v této práci, bude platforma od AVI networks. Narozdíl od HAproxy není tato platforma standardní součástí OpenStacku. Z tohoto důvodu musela být nejprve do testovacího prostředí nainstalována.

Instalace spočívá ve vytvoření tenantu, ve kterém je následně vytvořena virtuální instance z image pro AVI Controller. Této instanci je následně nutné přidat IP adresu, aby k ní bylo možné přistoupit zkrze webový prohlížeč. Poté již následuje jednoduchá konfigurace, kde nutné správně nastavit připojení na keystone server. Tím je zajištěna integrace s OpenStackem.

Po úspěšné instalaci vznikla v OpenStacku další služba, která umožňuje vytváření load balancerů ve všech tenantech. Vše funguje tak, že uživatel většinou přes AVI Controller dashboard, kde se přihlásí pomocí uživatele v keystoneu, zvolí vytvoření virtuální služby, kde vybere servery a nastaví požadované parametry. Na základě těchto údajů controller vytvoří tzv. Service Engine, který provádí load balancing. Tento Service Engine však vznikne stále v tenantu určeném pro AVI networks, nikoliv v uživatelském tenantu. Obrázek č. 5.6 znázorňuje toto řešení.



Obrázek 5.6.: Znáznornění funkce AVI networks LbaaS, převzato z [40]

Přestože vytváření a ovládání LbaaS zkrze dashboard bylo otestováno a funguje, tak je zde příliš manuálních kroků. Avšak při analýze tohoto produktu bylo zjištěno, že AVI networks nově také poskytuje podporu pro vytváření potřebných zdrojů i pomocí heatu.

5.2.2.1. Heat template pro AVI networks

Pro tvorbu heat templatu pro AVI networks byl stejný template jako pro HA-proxy. S tím rozdílem, že byly nahrazeny části týkající se load balanceru. Pro lepší čitelnost zde tedy nebudou uvedeny části, které jsou v obou templatech a budou zde popsány pouze části, které se liší. Jsou to tyto dvě části:

- AVI network pool - Obsahuje základní parametry pro load balancer. Je především definice serverů resp. jejich ip adresu, mezi kterými bude následně

probíhat load balancing. Také je zde možná volba metody pro load balancing. Zde byla zvolena metoda round robin jako v předchozím příkladě.

```
pool:
  type: Avi::LBaaS::Pool
  properties:
    name: "testpool"
    default_server_port: 80
    health_monitor_uuids:
      - {get_resource: hm}
    lb_algorithm: LB_ALGORITHM_ROUND_ROBIN
  servers:
    - ip:
        addr: { get_attr: [myinstance1, first_address] }
        type: V4
        port: 80
    - ip:
        addr: { get_attr: [myinstance2, first_address] }
        type: V4
        port: 80
```

Ukázka kódu 5.4: AVI networks pool

- AVI virtual service - Zde je definovaná především veřejná ip adresa a port. Přes tyto údaje se následně bude přistupovat k serverů, resp. webové aplikaci.

```
vs:
  type: Avi::LBaaS::VirtualService
  properties:
    name: "testvs"
    pool_uuid: {get_resource: pool}
    ip_address:
      addr: 10.10.32.100
```

```
type: V4
services:
  - port: 80
application_profile_uuid:
  get_avi_uuid_by_name: System-Secure-HTTP
```

Ukázka kódu 5.5: AVI networks health monitor

- AVI network health monitor - Poslední částí je health monitor pro sledování stavu jednotlivých serverů. V ukázce je vidět nastavení jednotlivých časovačů a také protokolu http. Monitor sleduje http odpověď od serveru a pokud dostane odpověď začínající 2xx či 3xx, tak označí server/aplikaci jako v pořádku a bude na ni zasílat data.

```
hm:
  type: Avi::LBaaS::HealthMonitor
  properties:
    name: "mytesthm"
    receive_timeout: 2
    failed_checks: 2
    successful_checks: 6
    send_interval: 2
    type: HEALTH_MONITOR_HTTP
    http_monitor:
      http_response_code:
        - HTTP_2XX
        - HTTP_3XX
      http_request: "GET / HTTP/1.0"
```

Ukázka kódu 5.6: AVI networks virtual service

Takto by tedy měla být vytvořena automatizovaná VNF využívající platformu od firmy AVI networks. Nyní ji zbývá otestovat.

5.2.2.2. Testování heat templatu pro AVI networks

Při testování heat templatu pro AVI networks bylo očekáváno, že zde bude dosaženo stejného stavu jako při použití dashboardu. Bohužel po spuštění daného templatu nedošlo k vytvoření požadovaných resourců a jeho běh skončil errorem. Při dalším zkoumání bylo zjištěno, že v době psaní této práce je podpora heat ze strany AVI networks příliš nová (přibližně měsíc) a není zde vše naprosto funkční. Z tohoto důvodu nemohl být tento heat template otestován.

5.3. Realizace VNF pro Fwaas

V této sekci jsou vysvětleny jednotlivé kroky související s realizací VNF pro Fwaas. Oproti předchozímu příkladu zde není přímá integrace s OpenStackem. Z tohoto důvodu bylo navrženo trochu odlišné řešení. Nejprve bude vytvořena virtuální infrastruktura pomocí heat templatu a tzv. Servisních instancí v OpenContrailu. Toto se nebude příliš od předchozího případu. Avšak poté bude samostatně řešen management resp. dodání potřebné konfigurace do vytvořené instance pro firewall. Tato konfigurace musí mít všechno potřebné, aby firewall vykonával funkci NAT, která byla zvolena jako ukázková VNF v této práci.

5.3.1. Servisní instance v OpenContrailu

V OpenContrailu je sice možnost využívat implementaci routeru s SNAT, která umožňuje instancím v privátních sítích konektivitu s externí sítí. Pokud však uživatel potřebuje využít pokročilejší funkce firewallu, tak je možné vytvořit servisní instanci, která bude sloužit jako VNF. V té může být použit libovolný požadovaný image firewallu uživatele.

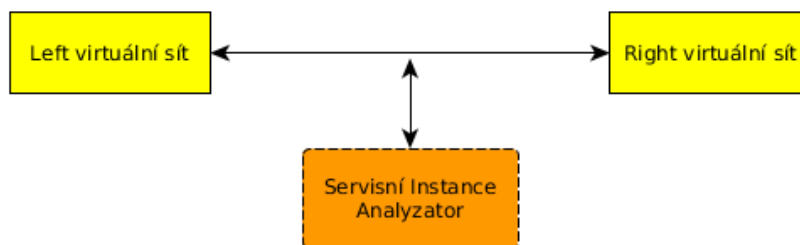
Dle informací uvedených v [30], tak servisní instance v OpenContrailu je jednoduše virtuální stroj, který poskytuje danou VNF. Zároveň je však možné využívat Service Chainingu, který byl popsán v kapitole č. 2. Úplně nejjednodušším příkladem může být virtuální stroj s operačním systémem GNU/Linux, který může sloužit jako router mezi dvěma sítěmi.

Pro vytvoření servisní instance je nejprve určit Servisní template, ze kterého má být tato instance vytvořena. Ten obsahuje její obecný předpis, který má tyto parametry:

- **Název** - Název je označení daného Servisního Templatu. Pomocí něho lze následně identifikovat daný template a spustit dle jeho parametrů Servisní instanci.
- **Image** - Je image, který má být použit pro vytvoření dané servisní instance. V našem případě se bude jednat o image, který obsahuje požadované síťové funkce. Tento image musí před tím než může být použit nahrán do OpenStacku Glance.
- **Service Type** - V OpenContrailu, prozatím existují dva typy. Jsou to Trafic Analyzer a Firewall.
- **Service Mode** - Zde se určuje v jakém modu daný template bude nastaven. Jsou zde 3 možnosti. , .
 - **Transparent** - v tomto případě se jedná o neroutovaný firewall, neboli L2 firewall.
 - **In-Network** - poskytuje výchozí bránu a průchozí traffic je routovaný. Tento mode může být využit pro NAT, HTTP proxy, atd.
 - **In-Network-NAT** - zde je situace podobná jako u In-Network, ale navracující traffic nemusí být routovaný zpět do zdrojové sítě.
- **Typy síťových portů** - Zde se určuje kolik portů bude daná instance, vytvořená pomocí tohoto templatu mít a jaká bude jejich role. Jsou zde možnosti Left, Right a Management.

Po úspěšném vytvoření Servisního templatu je možné z něj vytvořit libovolný počet Servis Instancí. Ty běží jako klasické instance v OpenStacku. Jak tedy vyplývá z výše uvedených informací, tak existují dva druhy servisních instancí v OpenContrailu.

První z nich je Analyzer. Ten slouží k analýze a zachytávání síťového trafficu. Image pro tento typ servisní instance obvykle obsahuje protokolový analyzátor a paketový sniffer, jako je například oblíbený program Wireshark. Tato instance dostává traffic, který je posílán mezi dvěma sítěmi. Tento traffic vybírá OpenContrail podle nastaveného pravidla pro dané sítě. Podle těchto pravidel je vybrána jen část trafficu, která je následně dána k dispozici servisní instanci. Samotná servisní instance nijak nemanipuluje s trafficem a ani do něj žádný negeneruje. Jednoduše lze říci, že má nastavený síťový port v promiskuitním modu a pouze pozoruje traffic. Poté jen hlásí zachycené události uživateli či jiným entitám v síti. Obrázek č. 5.7 znázorňuje tento typ servisní instance.

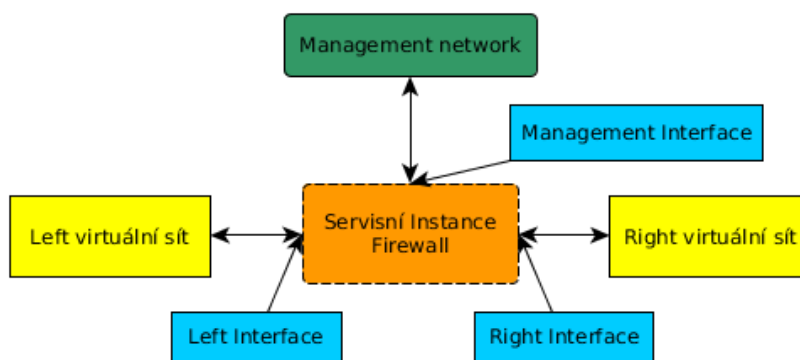


Obrázek 5.7.: Schéma zapojení servisní instance Analyzer

Je zde zřejmé, že pro účely této práce není Analyzer potřebný. Z tohoto důvodu bude dále uvažován pouze druhý typ servisní instance, kterým je Firewall. V tomto případě již servisní instance manipuluje s trafficem. Hlavní bodem, při vytváření servisní instance jako firewall, je přiřadit správné virtuální sítě k správným virtuálním síťovým portům. Servisní instance má obvykle dva síťové porty - left a right. Ty slouží pro propojení sítí do kterých jsou zapojeny. V některých případech je možné servisní instanci přidat i třetí síťový port, který slouží pro out-of-band management. Tento interface je vhodný především pro manuální management či pro management pomocí nějaké další entity.

Pro procházení síťové komunikace mezi sítěmi je nutné jí explicitně povolit v OpenContrailu pomocí tzv. Servisní policy. V tomto případě jich bude využito k vynucení toho, aby byla data posílána skrze servisní instanci.

Shrňme-li co vše je potřebné vytvořit pro FwaaS, tak je to servisní template,



Obrázek 5.8.: Schéma zapojení servisní instance

servisní instance z toho templatu a servisní policy. Toto vše lze naštěstí opět definovat v heat template, takže je možné toto všechno vytvořit automaticky jako v předchozím případě.

5.3.2. Heat template pro Fwaas

Heat template pro Fwaas byl navrhnout obdobně jako v případě Lbaas. Vzhledem k tomu, že template opět značně dlouhý, tak jeho kompletní verzi lze nalézt v příloze. Tam lze také nalézt enviroment file pro tento template. Zde bude uvedeno pouze to podstatné. Jeho součásti tedy jsou:

- virtuální instance pro testování - Tato instance budou sloužit pro otestování správné funkčnosti VNF. Její hlavní činností bude generování dat v síti.
- privátní síť - V této síti bude vytvořena instance pro testování. K této síti bude také připojen jeden z interfaců virtuální instance.
- firewall template - Zde je ukázka pro vytvoření servisního tempaltu, který je použit v této práci. Většina parametrů by měla být dle názvu. Jediné co může být nejasné je paramenter "service interface type list". Do tohoto paramentru jsou v enviroment file vloženy názvy a pořadí jednotlivých interfaců. V našem případě používáme "management,right,left".

```
service_template:
```

```
type: OS::Contrail::ServiceTemplate
properties:
  name: test-fw-template
  service_mode: in-network-nat
  service_type: firewall
  image_name: { get_param: template_image }
  service_scaling: True
  availability_zone_enable: True
  ordered_interfaces: { get_param: ordered_interfaces }
  flavor: { get_param: template_flavor }
  service_interface_type_list: { "Fn::Split" : [ ",",
    Ref: service_interface_type_list ] }
  shared_ip_list: { "Fn::Split" : [ ",", Ref:
    shared_ip_list ] }
  static_routes_list: { "Fn::Split" : [ ",", Ref:
    static_routes_list ] }
```

Ukázka kódu 5.7: Servisní template

- firewall instance - Tady je ukázán předpis pro tvorbu servisní instance. Je zde vidět, že instance bude vytvořena z již popsaného servisního template. Dále uveden seznam síťových interfaců společně s tím, kam se mají napojit. První se připojí do defaultní management sítě, kterou automaticky vytvoří OpenContrail, druhý do veřejné sítě a třetí do privátní.

```
service_instance:
  type: OS::Contrail::ServiceInstance
  depends_on: [private_subnet_1]
  properties:
    name: { get_param: private_instance_name }
    service_template: { get_resource: service_template}
    availability_zone: { get_param:
      private_availability_zone}
```

```
scale_out:
  max_instances: 1
interface_list: [
  {
    virtual_network: "auto"
  },
  {
    virtual_network: {get_param: public_net}
  },
  {
    virtual_network: {get_resource: private_net_1}
  }
]
```

Ukázka kódu 5.8: Servisní instance

- **contrail policy** - Poslední co bylo nutné definovat pro Fwaas je servisní policy. Zde bylo pro jednoduchost povoleno vše z obou směrů pro servisní instance. Tím bude jasné, že výsledný datový tok ovlivňuje pouze sdaná VNF.

```
private_policy:
  type: OS::Contrail::NetworkPolicy
  depends_on: [ private_net_1, service_instance ]
  properties:
    name: { get_param: policy_name }
    entries:
      policy_rule: [
        {
          "direction": "<>",
          "protocol": "any",
          "src_ports": [{ "start_port": -1, "end_port":
            -1}],
          "dst_ports": [{ "start_port": -1, "end_port":
            -1}],
        }
      ]
  }
```



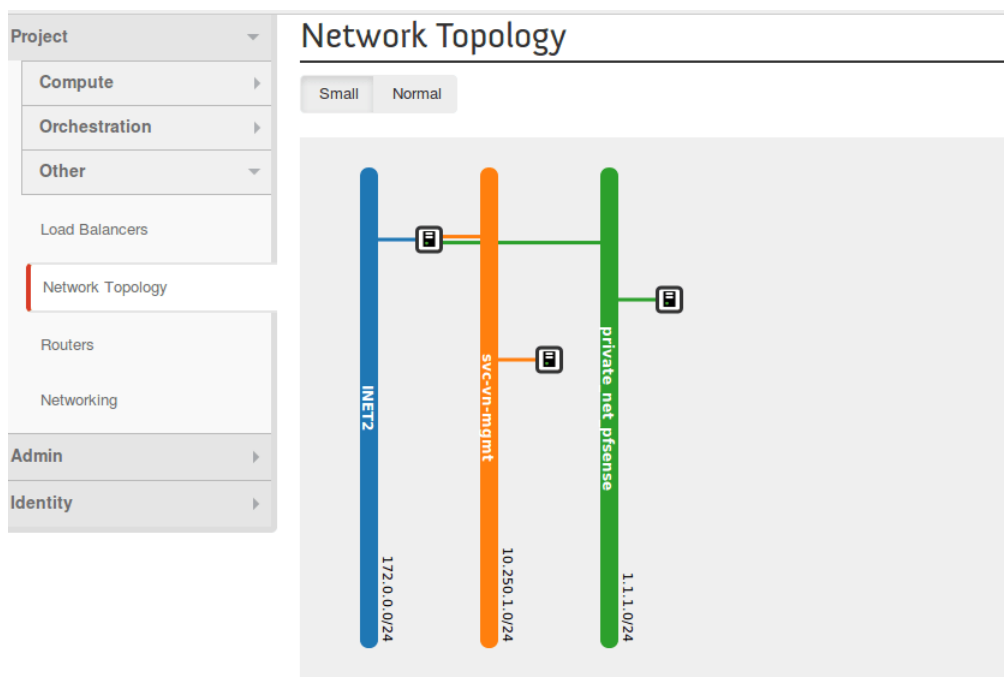
```

    "dst_addresses": [{ "virtual_network":
        {get_param: public_net}}],
    "action_list": {"apply_service":
        [{get_resource: service_instance}]},
    "src_addresses": [{ "virtual_network":
        {get_resource: private_net_1}}]
    },
]

```

Ukázka kódu 5.9: Contrail network policy

Tento template lze použít pro oba testované firewally. Jediné v čem se budou lišit jsou vstupní parametry, pro které budou dva oddělené environment soubory. Po spuštění daného templatu se vytvoří topologie, kterou znázorňuje obrázek č. 5.9.



Obrázek 5.9.: Síťová topologie

Přestože výsledná síťová topologie je stejná pro PfSense i pro Fortigate VM, tak je dále nutné zajistit jejich správnou konfiguraci. Ta však dle bližším prozkoumání

možností obou řešení bude značně odlišná. Proto již budou tyto informace uveřejněny odděleně pro každý z obou firewallů. Je nutné ještě zmínit, že instance, ze které bude probíhat management, není součástí výše zmíněného templatu. Tato instance byla vytvořena samostatně. Je to z důvodu toho, že management nástroje by byly neustále smazávány při testování templatu a bylo by nutné nalézt mechanismus, jak je vždy dostat zpět.

5.3.3. Management PfSense

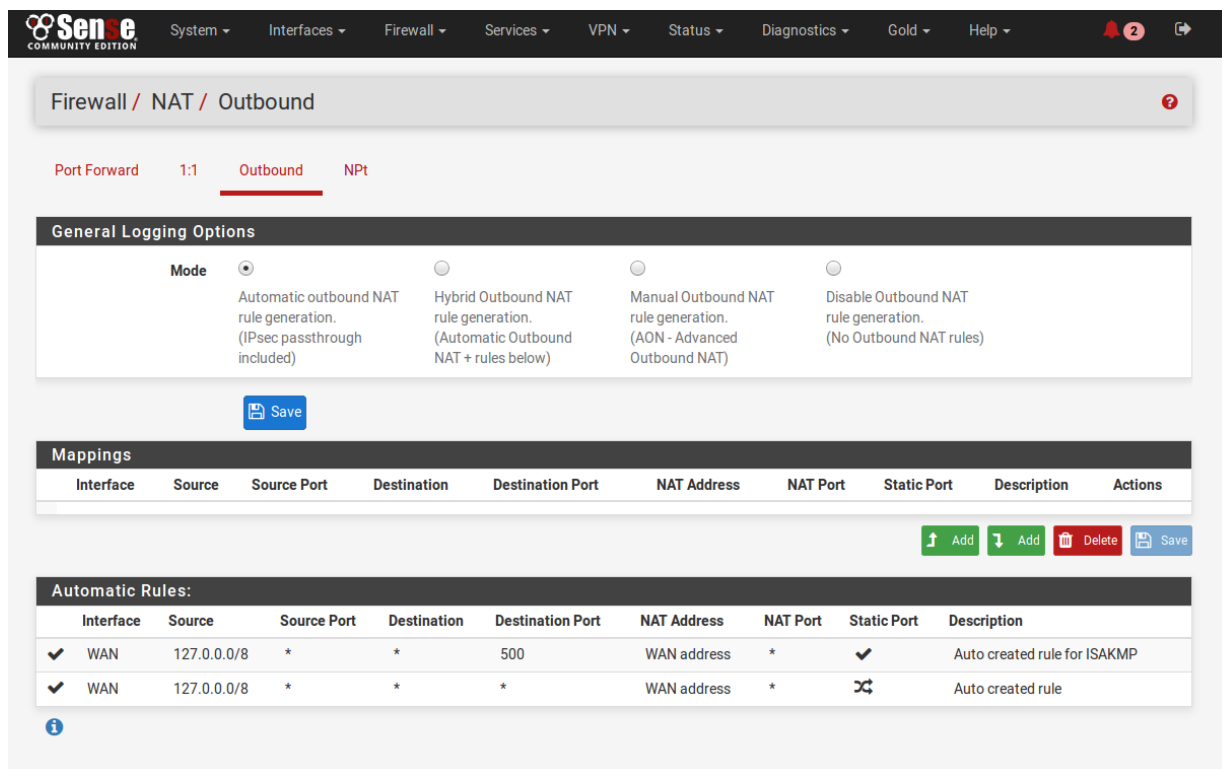
Pro management PfSense bylo oddzkoušeno několik možností. Je tomu tak, protože se ukázalo, že není úplně snadné proces konfigurace s PfSense elegantně vyřešit.

Vzhledem k tomu, že PfSense je postavený na operačním systému FreeBSD, tak úplně prvním uvažovaným řešením měl být skript, který by byl součástí heat templatu. K tomu by však bylo nutné, aby jeho součástí byl tzv. cloud-init, který by tento skript z heat templatu převzal a provedl. Jeho nainstalování do PfSense bohužel bylo neúspěšné.

Druhým přístupem bylo využití konfiguračního managementu SaltStack. Informace o tomto nástroji, lze nalézt zde [44]. Pro použití SaltStacku je nutné využít dvou jeho částí. První je tzv. Salt master. Ten obsahuje definici veškeré konfigurace pro infrastrukturu, kterou spravuje. Ten je v námi uvažovaném případě nainstalovaný na management instanci a obsahuje potřebné informace pro správnou konfiguraci VNF. Druhou částí je tzv. Salt minion, který má za úkol přijímat příkazy od salt mastera a ty následně na cílové instanci definovaným způsobem vynutit. Avšak zde nastává problém, protože se opět nepodařilo správně nainstalovat salt minionu do PfSense instance.

Po předchozích neúspěších byl tedy zvolen nejjednodušší možný způsob. Tím je přednastavení požadované konfigurace do image s PfSense. Toto se ukázalo jako optimální řešení. Je tomu tak, protože tento image musel být stejně předem ručně vytvořen ze staženého instalačního souboru a následně nahrán do prostředí OpenStack, kde mohl být poté využíván. Předkonfigurování proběhlo přes Pf-

Sense dashboard, jehož ukázkou ilustruje obrázek č. 5.10



Obrázek 5.10.: Konfigurace NAT v PfSense dashboardu

Vše muselo být uspůsobeno tak, aby image byl co nejvíce generický. To znamená, že na všech interfacech bylo nastaveno dhcp, protože uživatel může zvolit naprosto jiné ip rozsahy než jsou používány v této práci. To samé platí pro pravidla, které PfSense využívá pro NAT. Ty museli počítat s tím, že na interfacech budou různé ip adresy, takže se v nich nesměla konkrétní ip adresa vyskytovat. Toho bylo dosaženo tak, že v pravidle pro NAT byla ip adresa resp. LAN subnet, který by je jinak specificky definovaný, byl nahrazen čtyřmi nulami (0.0.0.0). Tak bylo dosaženo dosaženo správného překladu adres, bez ohledu na přidělené ip adresy z OpenContrailu.

Toto řešení muselo být samozřejmě řádně otestováno. Test proběhl pomocí příkazu ping z testovací instance. Jak je vidět na obrázku č. 5.11, tak konektivita do externí sítě funguje.

```

root@test-firewall:~#
root@test-firewall:~#
root@test-firewall:~#
root@test-firewall:~#
root@test-firewall:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=9.76 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=9.12 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=116 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=55 time=61.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=55 time=58.1 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 9.129/50.956/116.444/39.733 ms
root@test-firewall:~#

```

Obrázek 5.11.: Test konektivity zkrze PfSense

5.3.4. Management Fortigate VM

Management Fortigate VM probíhal obdobně jako u PfSense. Nejprve proběhlo hledání možností konfigurace zkrze skript v heat templatě. Zde však chybí jakákoli podpora doinstalace softwaru třetích stran, takže opět nemohl být použit cloud-init. Další hledání však objevilo, že Fortigate VM poskytuje API, které může být využito pro jeho management. K tomuto účelu bylo využito python knihovny a dokumentace [45].

Celé toto řešení funguje tak, že je nejprve vytvořena potřebaná konfigurace jako textový soubor a ta se následně pomocí python skriptu nahraje do Fortigate VM instance. V následující ukázce je příklad konfigurace pro síťový interface1. Takto lze ovládat všechny porty, kterými zařízení disponuje. Nejdůležitější se opět ukázalo povolení dhcp na používaných interfacech.

```

config system interface
edit port1
set type physical
set vdom root
set mode dhcp
set defaultgw disable
set allowaccess ssh ping http https
end

```

Ukázka kódu 5.10: Fortigate konfigurace pro interface

Dále bylo nutné definovat konfiguraci pro NAT. V té bylo nastaven překlad adres z interfacu 3 na interface 2.

```
config firewall policy
edit 1
set srcintf port3
set dstintf port2
set srcaddr all
set dstaddr all
set action accept
set schedule always
set service ALL
set logtraffic all
set nat enable
end
```

Ukázka kódu 5.11: Fortigate konfigurace pro NAT

Obě tyto konfigurace následně mohli být nahrány do Fortigate VM pomocí následujícího python skriptu. Ten se přihlásí do zařízení pomocí poskytnutých přihlašovacích údajů a stáhne si aktuální konfiguraci. V tomto případě konfiguraci pro všechny interfaci. Tu následně poroná s námi poskytnutou konfigurací a v případě změny odešle zpět do zařízení. Python skript pro nahrání konfigurace pro NAT je téměř stejný. Pouze otevírá jiný soubor a porovnává ho s konfigurací s sekci "firewall policy".

```
from pyFG import FortiOS
import sys
if __name__ == '__main__':
    f = open('fortios_intf.txt', 'r')
    candidate = f.read()
    f.close()
    d = FortiOS('fortigate', username="admin",
                password="fortigate")#, vdom='test_vdom')
```

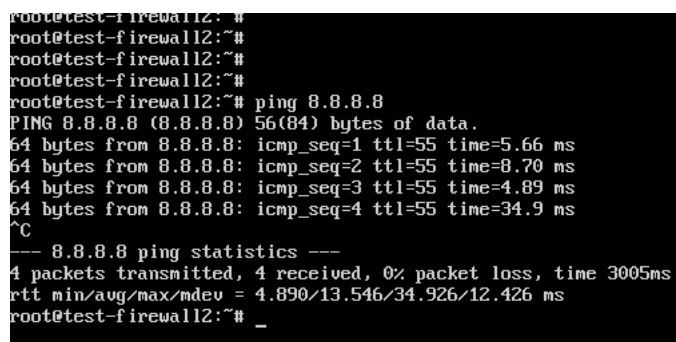
```

d.open()
d.load_config('system interface', empty_candidate=True)
d.load_config(config_text=candidate, in_candidate=True)
config_changes = d.compare_config()
d.commit(config_changes)
d.close()

```

Ukázka kódu 5.12: Skript pro nahrání konfigurace interfaců

Stejně jako v předchozím případě byl nutný test konektivity po spuštění obou skriptů. Obrázek č. 5.12 ukazuje, že konektivita opět funguje. Je tedy zřejmé, že konfigurace VNF vytvořené pomocí Fortigate VM byla úspěšně nakonfigurována.



```

root@test-fw12:~#
root@test-fw12:~#
root@test-fw12:~#
root@test-fw12:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=5.66 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=8.70 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=4.89 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=55 time=34.9 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 4.890/13.546/34.926/12.426 ms
root@test-fw12:~# _

```

Obrázek 5.12.: Test konektivity zkrze Fortigate VM

5.4. Shrnutí získaných poznatků a zhodnocení

V této kapitole byly navrženo a otestováno několik příkladů, kterými lze přistupovat k managementu a orchestraci VNF. Zde bude uvedeno stručné shrnutí a zhodnocení těchto řešení. Zároveň budou zmíněny poznatky, které byly při jejich realizaci získány.

Z uvedených příkladů pro LbaaB je jasné, že pokud existuje přímá integrace s použitou platformou, tak management VNF může být velice jednoduché automatizovat zkrze již poskytnuté nástroje této platformy. Toto platí pro obě testované technologie.

Porovnáme-li mezi se sebou HAproxy a platformu AVI networks, tak zjistíme,

že obě řešení lze využít pro produkční použití. HAproxy je spíše určená pro uživatele, kteří potřebují jednodušší load balancer, který nemusí obsahovat velkou škálu konfigurace. Oproti tomu AVI networks je určená pro pokročilejší uživatele, kteří chtějí větší možnost konfigurace a monitorování. V obou případech byla provedena implementace automatického managementu. Bohužel v případě AVI networks tato implementace nemohla být otestována. Avšak obě implementace byly vytvořeny téměř stejným způsobem. Možnosti automatického managementu jsou tedy v obou případech srovnatelné.

V případě FwaaS je situace složitější. Jak pro Fortigate VM, tak pro PfSense byl použit stejný heat template. Lze tedy říci, že možnosti pro jejich automatické vytvoření jsou v obou případech stejné a souvisejí tedy s použitou platformou. V tomto případě byl zvolen OpenStack s OpenContrail. Díky využití OpenContrailu bylo možné využít tzv. Service chaining. Ten umožňoval jednoduché řízení datového toku zkrze obě řešení.

Avšak co se týče možností automatické konfigurace obou technologií, tak zde jsou již značné rozdíly, které souvisejí s podporou danou jejich poskytovateli. U obou případů bylo otestováno několik možností. Pro PfSense však nakonec muselo být zvoleno řešení v podobě předkonfigurovaného image. Tohoto řešení má výhody, především v tom, že se uživatel nemusí vůbec řešit konfiguraci. Pouze použít daný image a vše funguje tak, jak má. U Fortigate VM byla konfigurace doručena až po vytvoření celé virtuální infrastruktury. Pro toto řešení bylo využito API a python knihovna, která je k tomuto účelu poskytována. Zde výhodu značná flexibilita při konfiguraci. Je totiž možné ji měnit v průběhu celého životního cyklu instance. Oproti tomu je zde nevýhodou nutnost spustit skript manuálně po vytvoření virtuální infrastruktury.

6. Závěr

Virtualizace síťových funkcí je dnes jedno z aktuálních témat v oblasti počítačových sítí a IT světě. Je to z důvodů zvyšujících se požadavků na počítačové sítě, která je zapříčiněna rychlým vývojem a nasazováním nových technologií jako je např. cloud computing. Dalším důvodem jsou stoupající požadavky společností a telekomunikačních poskytovatelů síťových služeb na efektivnější a flexibilnější provoz počítačové sítě.

Tato práce si dala za cíl prověřit a vyzkoušet možnosti managementu a orchestrace virtuálních síťových služeb. V práci byla zmíněna základní problematika NFV a zároveň bylo realizováno několik příkladů VNF, na kterém byly demonstrovány jejich aktuální možnosti automatizovaného managementu. Z výsledků práce lze usoudit, že aktuálně existují nástroje, které dokáží zajistit automatickou konfiguraci VNF, avšak je nutné zajistit řádnou podporu ze strany poskytovatelů těchto řešení.

Jako vhodný předmět pro další výzkum je možné označit integraci jednotlivých management nástrojů přímo cloudové platformy. Tím by byly zajištěny lepší možnosti automatizace. Dále je také možné označit za vhodný předmět k dalšímu výzkumu i podporu výběru fyzického umístění jednotlivých VNF. Tím by mohl být výrazně obnoven jejich výkon.

Literatura

- [1] WEINS, Kim. *Cloud Computing Trends: 2016 State of the Cloud Survey*. In: RightScale [online]. 2016 [cit. 2016-08-13]. Dostupné z: <http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey>
- [2] STEVENSON, Rick. *How Low-Cost Telecom Killed Five 9s in Cloud Computing*. In: Wired [online]. 2013 [cit. 2016-08-12]. Dostupné z: <http://www.wired.com/insights/2013/03/how-low-cost-telecom-killed-five-9s-in-cloud-computing/>
- [3] SKOLDSTROM, Pontus, Balazs SONKOLY, Andras GULYAS, et al. Towards Unified Programmability of Cloud and Carrier Infrastructure. In: 2014 Third European Workshop on Software Defined Networks [online]. IEEE, 2014, s. 55-60 [cit. 2016-08-12]. DOI: 10.1109/EWSDN.2014.18. ISBN 978-1-4799-6919-7. Dostupné z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6984052>
- [4] R. Guerzoni, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action. Introductory white paper," in SDN and OpenFlow World Congress, June 2012. [online]. [cit. 2016-04-07]. Dostupné také z: https://portal.etsi.org/NFV/NFV_White_Paper.pdf
- [5] HAN, Bo, Vijay GOPALAKRISHNAN, Lusheng JI a Seungjoon LEE. *Network function virtualization: Challenges and opportunities for innovations*. IEEE Communications Magazine. 2015, 53(2), 90-97. DOI:

- 10.1109/MCOM.2015.7045396. ISSN 0163-6804. Dostupné také z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7045396>
- [6] MIJUMBI, Rashid, Joan SERRAT, Juan-Luis GORRICHIO, Niels BOUTEN, Filip DE TURCK a Raouf BOUTABA. *Network Function Virtualization: State-of-the-Art and Research Challenges*. IEEE Communications Surveys. 2016, 18(1), 236-262. DOI: 10.1109/COMST.2015.2477041. ISSN 1553-877x. Dostupné také z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7243304>
- [7] KUSNETZKY, Dan. *Virtualization: a manager's guide*. Sebastopol, CA: O'Reilly, c2011. ISBN 1449306454.
- [8] J. Smith and R. Nair, *The architecture of virtual machines*, Computer, vol. 38, no. 5, pp. 32–38, May 2005. doi: 10.1109/MC.2005.173
- [9] MELL, Peter a Tim GRANCE. *The NIST definition of cloud computing*. [online]. In: . 2011 [cit. 2016-08-13]. Dostupné z: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>
- [10] FONSECA, Nelson L. S. da. a Raouf BOUTABA. *Cloud services, networking, and management*. Hoboken, New Jersey: Wiley, 2015. ISBN 9781118845943.
- [11] SHERRY, Justine, Shaddi HASAN, Colin SCOTT, Arvind KRISHNAMURTHY, Sylvia RATNASAMY a Vyas SEKAR. *Making middleboxes someone else's problem*. In: Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication - SIGCOMM '12 [online]. New York, New York, USA: ACM Press, 2012, s. 13- [cit. 2016-08-07]. DOI: 10.1145/2342356.2342359. ISBN 9781450314190. Dostupné z: <http://dl.acm.org/citation.cfm?doid=2342356.2342359>
- [12] KREUTZ, Diego, Fernando M. V. RAMOS, Paulo ESTEVES VERISSIMO, Christian ESTEVE ROTHENBERG, Siamak AZODOLMOLKY a Steve UH-

- LIG. *Software-Defined Networking: A Comprehensive Survey*. Proceedings of the IEEE [online]. 2015, 103(1), 14-76 [cit. 2016-04-09]. DOI: 10.1109/J-PROC.2014.2371999. ISSN 0018-9219. Dostupné z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6994333>
- [13] WOOD, Timothy, K. K. RAMAKRISHNAN, Jinho HWANG, Grace LIU a Wei ZHANG. Toward a software-based network: integrating software defined networking and network function virtualization. IEEE Network [online]. 2015, 29(3), 36-41 [cit. 2016-08-07]. DOI: 10.1109/MNET.2015.7113223. ISSN 0890-8044. Dostupné z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7113223>
- [14] DOHERTY, Jimmy. *SDN and NFV simplified: a visual guide to understanding software defined networks and network function virtualization*. 1st edition. Indianapolis, IN: Addison-Wesley Professional, 2016. ISBN 9780134306407.
- [15] ETSI Industry Specification Group (ISG) NFV, “ETSI GS NFV 002 V1.2.1: Network Functions Virtualisation (NFV); Architectural Framework,” December 2014. [online]. [cit. 2016-04-07]. Dostupné také z: <http://www.etsi.org/deliver/etsigs/NFV/001099/002/01.02.0160/gsNFV002v010201p.pdf>
- [16] ETSI Industry Specification Group (ISG) NFV, “ETSI GS NFV 003 V1.2.1: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV,” December 2014. [online]. [cit. 2016-04-07]. <http://www.etsi.org/deliver/etsigs/NFV/001099/003/01.02.0160/gsNFV003v010201p.pdf>
- [17] ETSI Industry Specification Group (ISG) NFV, *ETSI GS NFV-INF 001 V1.1.1: Network Functions Virtualisation (NFV);Infrastructure Overview* , 2015. [online]. [cit. 2016-04-07].http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_nfv-inf001v010101p.pdf

-
- [18] ETSI Industry Specification Group (ISG) NFV, *ETSI GS NFV-INF 003 V1.1.1: Network Functions Virtualisation (NFV); Infrastructure; Compute Domain*, 2014. [online]. [cit. 2016-04-07]. http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/003/01.01.01_60/gs_NFV-INF003v010101p.pdf
- [19] ETSI Industry Specification Group (ISG) NFV, *ETSI GS NFV-INF 004 V1.1.1: Network Functions Virtualisation (NFV); Infrastructure; Hypervisor Domain*, 2015. [online]. [cit. 2016-01-07]. http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/004/01.01.01_60/gs_nfv-inf004v010101p.pdf
- [20] ETSI Industry Specification Group (ISG) NFV, *ETSI GS NFV-INF 005 V1.1.1: Network Functions Virtualisation (NFV); Infrastructure; Network Domain*, 2014. [online]. [cit. 2016-03-05]. http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/005/01.01.01_60/gs_NFV-INF005v010101p.pdf
- [21] ETSI Industry Specification Group (ISG) NFV, *ETSI GS NFV-SWA 001 V1.1.1: Network Functions Virtualisation (NFV); Virtual Network Functions Architecture*, 2014. [online]. [cit. 2016-02-09]. http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_nfv-swa001v010101p.pdf
- [22] ETSI Industry Specification Group (ISG) NFV, *ETSI GS NFV-MAN V1.1.1: Network Functions Virtualisation (NFV); Management and Orchestration*, 2014. [online]. [cit. 2016-02-01]. http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf
- [23] MIJUMBI, Rashid, Joan SERRAT, Juan-luis GORRICO, Steven LATRE, Marinos CHARALAMBIDES a Diego LOPEZ. *Management and orchestration challenges in network functions virtualization*. IEEE Communications Magazine.

- 2016, 54(1), 98-105. DOI: 10.1109/MCOM.2016.7378433. ISSN 0163-6804.
Dostupné také z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7378433>
- [24] Openstack: Open source cloud computing software. 2016.[Online]. OpenStack Foundation, © 2016 [cit. 2016-08-13] Dostupné z: <https://www.openstack.org>
- [25] KHEDHER, Omar. *Mastering OpenStack: design, deploy, and manage a scalable OpenStack infrastructure*. First published. Birmingham: Packt Publishing, 2015. Community experience distilled (Packt). ISBN 978-1-78439-564-3.
- [26] Heat; OpenStack Orchestration. [online]. OpenStack Foundation, © 2016 [cit. 2016-08-13] Dostupné z: <https://wiki.openstack.org/wiki/Heat>
- [27] OpenStack User Survey. [online]. OpenStack Foundation, 2016 [cit. 2016-08-13]. Dostupné z: <https://www.openstack.org/assets/survey/April-2016-User-Survey-Report.pdf>
- [28] Open Platform for NFV (OPNFV) [online]. Open Platform for NFV Project, Inc., © 2016 [cit. 2016-08-14]. Dostupné z: <https://www.opnfv.org/>
- [29] RIJSMAN, Bruno a Ankur SINGLA. *Day One: Understanding OpenContrail Architecture*. Juniper Networks Books, 2013
- [30] OpenContrail. *OpenContrail Architecture Documentation*. [online]. 2016 [cit. 2016-08-10]. Dostupné z: <http://www.opencontrail.org/opencontrail-architecture-documentation/>
- [31] Virtualizace od společnosti VMware pro desktopy, servery, aplikace, veřejné a hybridní cloudy [online]. VMware, Inc., 2016 [cit. 2016-08-14]. Dostupné z: <http://www.vmware.com/cz.html>
- [32] GALLAGHER, Simon a Aidan DALGLEISH. *VMware private cloud computing with vCloud Director*. Indianapolis, Ind.: Sybex, c2013

- [33] Let Us GO Virtual VMware vCloud Suite. *What is it? What does it mean to VMware Customers*. [online]. 2012 [cit. 2016-08-01] Dostupné z: <http://letusgovirtual.com/?p=591>
- [34] ITProPortal. *The top enterprise firewalls of 2015*. [online]. 2015 [cit. 2016-07-01]. Dostupné z: <http://www.itproportal.com/2015/08/21/the-top-enterprise-firewalls-of-2015/>
- [35] Fortinet. *Fortinet - Virtual Appliances*. [online]. 2016 [cit. 2016-08-01]. Dostupné z: <https://www.fortinet.com/products-services/products/firewall/fortigate-virtual-appliances.html>
- [36] Juniper Networks. *VSRX Integrated Virtual Firewall*. [online]. 2016 [cit. 2016-08-02]. Dostupné z: <http://www.juniper.net/us/en/products-services/security/srx-series/vsrx/>
- [37] Cisco. *Cisco Adaptive Security Virtual Appliance (ASAv)*. 2016 [online]. [cit. 2016-08-02]. Dostupné z: <http://www.cisco.com/c/en/us/products/security/virtual-adaptive-security-appliance-firewall/index.html>
- [38] PfSense. *PfSense Overview*. [online]. 2016 [cit. 2016-08-02]. Dostupné z: <https://www.pfsense.org>
- [39] HAProxy *HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer*. [online]. 2016 [cit. 2016-08-02]. Dostupné z: <http://www.haproxy.org/>
- [40] AVI networks *AVI networks - Load Balancing as a Service for OpenStack*. [online]. 2016 [cit. 2016-08-02]. Dostupné z: <https://avinetworks.com/load-balancing-as-a-service-for-openstack/>
- [41] MOENS, Hendrik a Filip De TURCK. VNF-P: A model for efficient placement of virtualized network functions. In: 10th International Conference on Network and Service Management (CNSM) and Workshop [online]. IEEE, 2014, s. 418-423 [cit. 2016-08-14]. DOI: 10.1109/CNSM.2014.7014205.

- ISBN 978-3-901882-67-8. Dostupné z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7014205>
- [42] CLEMM, Alexander. *qNetwork management fundamentals*. Indianapolis, IN: Cisco Press, c2007. ISBN 1587201372.
- [43] SMOLA, Ondřej. *Automatizace síťového provozu na operačních systémech směrovačů*. Bakalářská práce. Univerzita Hradec Králové, Fakulta Informatiky a Managementu. Vedoucí práce Vladimír Soběslav.
- [44] SaltStack *SaltStack automation for CloudOps, ITops and DevOps at scale*. [online]. 2016 [cit. 2016-08-02]. Dostupné z: <https://saltstack.com/>
- [45] BARROSO, David. PyFG. *Welcome to pyFG's documentation!* [online]. 2014 [cit. 2016-08-06]. Dostupné z: <http://pyfg.readthedocs.io/en/latest/>

Seznam obrázků

2.1. Schéma hypervisorů	4
2.2. Tradiční schéma počítačové sítě	6
2.3. Koncept virtualizace síťových funkcí, převzato z [5]	6
2.4. Schéma SDN, převzato z [12]	8
2.5. Srovnání SDN a NFV	9
2.6. Ukázka klasického service chainigu pomocí fyzických síťových prvků	10
2.7. Ukázka VNF service chainigu	10
3.1. NFV architektura, převzato z [15]	13
3.2. Schéma NFV infrastruktury	14
3.3. Schéma virtuální síťové funkce	15
3.4. Schéma NFV MANO	17
3.5. Popis heat orchestrace	20
3.6. Schéma VMware vCloud Suite, převzato z [33]	23
4.1. Výsledná architektura NFV	31
4.2. Schéma výsledné fyzické topologie pro NFV	32
5.1. Firewall as a Service	34
5.2. Firewall as a Service	35
5.3. Neutron LbaaS	37
5.4. Vytvořená síťová topologie	40
5.5. Test konektivity a load balancingu	41
5.6. Znázornění funkce AVI networks LbaaS, převzato z [40]	42
5.7. Schéma zapojení servisní instance Analyzer	47

5.8. Schéma zapojení servisní instance	48
5.9. Síťová topologie	51
5.10. Konfigurace NAT v PfSense dashboardu	53
5.11. Test konektivity zkrze PfSense	54
5.12. Test konektivity zkrze Fortigate VM	56

Seznam tabulek

4.1. Srovnání VMware vCloud Suite a OpenStacku	29
4.2. Přehled softwaru pro LbaaS	30
4.3. Přehled softwaru pro FwaaS	31

Seznam ukázek kódu

5.1. Healt monitor	38
5.2. Healt monitor	38
5.3. Healt monitor	39
5.4. AVI networks pool	43
5.5. AVI networks healt monitor	43
5.6. AVI networks virtual service	44
5.7. Servisní template	48
5.8. Servisní instance	49
5.9. Contrail network policy	50
5.10. Fortigate konfigurace pro interface	54
5.11. Fortigate konfigurace pro NAT	55
5.12. Skript pro nahrání konfigurace interfaců	55

Přílohy

A. Heat template Lbaas - HAproxy

```
heat_template_version: 2013-05-23
description: LBAAS Template
parameters:
  key_name:
    type: string
  instance_flavor:
    type: string
    description: Instance type for servers
    default: m1.small
    constraints:
      - allowed_values: [m1.tiny, m1.small, m1.medium, m1.large]
        description: instance_type must be a valid instance type
  instance_image:
    type: string
    description: Image name to use for the servers.
    default: ubuntu-14-04-x64
  public_net_id:
    type: string
    description: ID or name of public network for which floating
      IP addresses will be allocated
  router_name:
    type: string
    description: Name of router to be created
    default: test-router
  lb_name:
```

```
    type: string
    description: Name of balancer to be created
    default: test-lb
public_net_name:
    type: string
    description: Name of public network to be created
    default: public-net
public_net_cidr:
    type: string
    description: Public network address (CIDR notation)
    default: 10.10.20.0/24
public_net_pool_start:
    type: string
    description: Start of public network IP address allocation
        pool
    default: 10.10.20.100
public_net_pool_end:
    type: string
    description: End of public network IP address allocation pool
    default: 10.10.20.200
private_net_name:
    type: string
    description: Name of private network to be created
    default: private-net
private_net_cidr:
    type: string
    description: Private network address (CIDR notation)
    default: 10.10.10.0/24
private_net_pool_start:
    type: string
    description: Start of private network IP address allocation
        pool
```

```
    default: 10.10.10.100
private_net_pool_end:
  type: string
  description: End of private network IP address allocation
    pool
    default: 10.10.10.200
resources:
  http_security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      name: http
      rules:
        - direction: ingress
          remote_mode: remote_ip_prefix
          remote_ip_prefix: 0.0.0.0/0
          port_range_min: 80
          port_range_max: 80
          protocol: tcp
  public_net:
    type: OS::Neutron::Net
    properties:
      admin_state_up: True
      name: { get_param: public_net_name }
      shared: False
  public_subnet:
    type: OS::Neutron::Subnet
    properties:
      allocation_pools:
        - start: { get_param: public_net_pool_start }
          end: { get_param: public_net_pool_end }
      cidr: { get_param: public_net_cidr }
      enable_dhcp: True
```

```
    ip_version: 4
    name: { get_param: public_net_name }
    network_id: { get_resource: public_net }
private_net:
  type: OS::Neutron::Net
  properties:
    admin_state_up: True
    name: { get_param: private_net_name }
    shared: False
private_subnet:
  type: OS::Neutron::Subnet
  properties:
    allocation_pools:
      - start: { get_param: private_net_pool_start }
        end: { get_param: private_net_pool_end }
    cidr: { get_param: private_net_cidr }
    enable_dhcp: True
    ip_version: 4
    name: { get_param: private_net_name }
    network_id: { get_resource: private_net }
router:
  type: OS::Neutron::Router
  properties:
    name: { get_param: router_name }
    external_gateway_info:
      network: { get_param: public_net_id }
router_interface:
  type: OS::Neutron::RouterInterface
  properties:
    router_id: { get_resource: router }
    subnet_id: { get_resource: private_subnet }
lb_ping_health_monitor:
```

```
type: OS::Neutron::HealthMonitor
properties:
  admin_state_up: True
  delay: 5
  max_retries: 1
  timeout: 5
  type: PING
lb_pool:
  type: OS::Neutron::Pool
  properties:
    admin_state_up: True
    lb_method: ROUND_ROBIN
    name: { get_param: lb_name }
    protocol: HTTP
    monitors:
      - { get_resource: lb_ping_health_monitor }
    subnet_id: { get_resource: private_subnet }
    vip:
      protocol_port: 80
#     address: { get_param: public_net_ip }
    admin_state_up: True
    subnet: { get_resource: public_subnet }
instance_01:
  type: OS::Nova::Server
  properties:
    image: { get_param: instance_image }
    flavor: { get_param: instance_flavor }
    key_name: { get_param: key_name }
    name: test-web01
    networks:
      - network: { get_resource: private_net }
    security_groups:
```

```
- default
- { get_resource: http_security_group }
user_data_format: RAW
user_data: |
    #!/bin/bash -v
    apt-get install apache2 -yy
    echo "Instance 01" > /var/www/html/index.html
lb_pool_member_instance_01:
  type: OS::Neutron::PoolMember
  properties:
    address: { get_attr: [ instance_01 , first_address ] }
    admin_state_up: True
    pool_id: { get_resource: lb_pool }
    protocol_port: 80
    weight: 1
instance_02:
  type: OS::Nova::Server
  properties:
    image: { get_param: instance_image }
    flavor: { get_param: instance_flavor }
    key_name: { get_param: key_name }
    name: test-web02
    networks:
      - network: { get_resource: private_net }
    security_groups:
      - default
      - { get_resource: http_security_group }
    user_data_format: RAW
    user_data: |
      #!/bin/bash -v
      apt-get install apache2 -yy
      echo "Instance 02" > /var/www/html/index.html
```

```
lb_pool_member_instance_02:
  type: OS::Neutron::PoolMember
  properties:
    address: { get_attr: [ instance_02 , first_address ] }
    admin_state_up: True
    pool_id: { get_resource: lb_pool }
    protocol_port: 80
    weight: 1
lb:
  type: OS::Neutron::LoadBalancer
  properties:
    members:
      - { get_resource: instance_01 }
      - { get_resource: instance_02 }
    pool_id: { get_resource: lb_pool }
    protocol_port: 80
lb_floating:
  type: OS::Neutron::FloatingIP
  properties:
    floating_network_id: {get_param: public_net_id}
    port_id: {get_attr: [lb_pool, vip, port_id]}
```

B. Enviroment file pro LbaaS - HAproxy

```
parameters:  
  instance_flavor: m1.small  
  instance_image: ubuntu-14-04-x64  
  key_name: rav_key  
  public_net_id: 22d00df7-b1ca-41a1-a615-89c3bdde6c10
```

C. Heat template FwaaS

heat_template_version: 2013-05-23

description: >

HOT template to creates two virtual network with one subnet
each.

Create a service instance

Creates a network policy [for](#) applying service between two VNs
created before.

Attach the network policy to two virtual networks

parameters:

policy_name:

type: string

description: Policy Name

direction:

type: string

description: Direction of Policy

start_src_ports:

type: number

description: Start of src port

end_src_ports:

type: number

description: End of src port

start_dst_ports:

type: number

description: Start of dst port

```
end_dst_ports:
  type: number
  description: End of dst port
private_net_1_name:
  type: string
  description: Name of private network to be created
private_net_1_cidr:
  type: string
  description: Private network address (CIDR notation)
private_net_1_gateway:
  type: string
  description: Private network gateway address
private_net_1_pool_start:
  type: string
  description: Start of private network IP address allocation
    pool
private_net_1_pool_end:
  type: string
  description: End of private network IP address allocation
    pool
public_net:
  type: string
  description: Name of private network to be created
private_instance_name:
  type: string
  default: TestService1
  description: service instance name
private_availability_zone:
  type: string
  default: ""
  description: availability zone in form of Zone:Host
max_instances:
```

type: number
description: maximum number of instances to be spawned

template_name:
type: string
description: Name of service template

template_mode:
type: string
description: service mode

template_type:
type: string
description: service type

template_image:
type: string
description: Name of the image

template_flavor:
type: string
description: Flavor

service_interface_type_list:
type: string
description: List of [interface](#) types

shared_ip_list:
type: string
description: List of shared ip enabled-disabled

static_routes_list:
type: string
description: List of [static](#) routes enabled-disabled

scaling:
type: string
description: Indicates whether service scaling is enabled

availability_zone:
type: string

```
    description: Indicates availability zone is enabled
ordered_interfaces:
    type: string
    description: Indicates service interfaces are ordered

key_name:
    type: string
instance_flavor:
    type: string
    description: Instance type for servers
    default: m1.small
    constraints:
        - allowed_values: [m1.tiny, m1.small, m1.medium, m1.large]
          description: instance_type must be a valid instance type
instance_image:
    type: string
    description: Image name to use for the servers.
    default: ubuntu-14-04-x64

resources:
    private_net_1:
        type: OS::Neutron::Net
        properties:
            name: { get_param: private_net_1_name }

    private_subnet_1:
        type: OS::Neutron::Subnet
        depends_on: private_net_1
        properties:
            network_id: { get_resource: private_net_1 }
            cidr: { get_param: private_net_1_cidr }
            gateway_ip: { get_param: private_net_1_gateway }
```

```
allocation_pools:
  - start: { get_param: private_net_1_pool_start }
    end: { get_param: private_net_1_pool_end }

service_instance:
  type: OS::Contrail::ServiceInstance
  depends_on: [private_subnet_1]
  properties:
    name: { get_param: private_instance_name }
    service_template: { get_resource: service_template}
    availability_zone: { get_param: private_availability_zone}
    scale_out:
      max_instances: { get_param: max_instances }
  interface_list: [
    {
      virtual_network: "auto"
    },
    {
      virtual_network: {get_param: public_net}
    },
    {
      virtual_network: {get_resource: private_net_1}
    }
  ]

private_policy:
  type: OS::Contrail::NetworkPolicy
  depends_on: [ private_net_1, service_instance ]
  properties:
    name: { get_param: policy_name }
    entries:
      policy_rule: [
```

```

    {
      "direction": { get_param: direction },
      "protocol": "any",
      "src_ports": [{ "start_port": {get_param:
        start_src_ports}, "end_port": {get_param:
        end_src_ports}}],
      "dst_ports": [{ "start_port": {get_param:
        start_dst_ports}, "end_port": {get_param:
        end_dst_ports}}],
      "dst_addresses": [{ "virtual_network":
        {get_param: public_net}}],
      "action_list": {"apply_service": [{get_resource:
        service_instance}]},
      "src_addresses": [{ "virtual_network":
        {get_resource: private_net_1}}]
    },
  ]

```

service_template:

type: OS::Contrail::ServiceTemplate

properties:

```

  name: { get_param: template_name }
  service_mode: { get_param: template_mode }
  service_type: { get_param: template_type }
  image_name: { get_param: template_image }
  service_scaling: { get_param: scaling }
  availability_zone_enable: { get_param: availability_zone }
  ordered_interfaces: { get_param: ordered_interfaces }
  flavor: { get_param: template_flavor }
  service_interface_type_list: { "Fn::Split" : [ ",", Ref:
    service_interface_type_list ] }
  shared_ip_list: { "Fn::Split" : [ ",", Ref: shared_ip_list

```

```
    ] }

    static_routes_list: { "Fn::Split" : [ ",", Ref:
        static_routes_list ] }

private_policy_attach_net1:
  type: OS::Contrail::AttachPolicy
  depends_on: [ private_net_1, private_policy ]
  properties:
    network: { get_resource: private_net_1 }
    policy: { get_attr: [private_policy, fq_name] }

private_policy_attach_net2:
  type: OS::Contrail::AttachPolicy
  depends_on: [private_policy ]
  properties:
    network: { get_param: public_net }
    policy: { get_attr: [private_policy, fq_name] }

test_instance_01:
  type: OS::Nova::Server
  properties:
    image: { get_param: instance_image }
    flavor: { get_param: instance_flavor }
    key_name: { get_param: key_name }
    name: test-web01
    networks:
      - network: { get_resource: private_net_1 }
    security_groups:
      - default
    user_data_format: RAW
    user_data: |
      #!/bin/bash -v
```

```
apt-get install apache2 -yy  
echo "Instance 01" > /var/www/html/index.html
```

D. Enviroment file pro LbaaS - Pfsense

```
parameters:
  private_instance_name: test-firewall
  max_instances: 1
  policy_name: contrail_policy1
  direction: "<>"
  start_src_ports: -1
  end_src_ports: -1
  start_dst_ports: -1
  end_dst_ports: -1

  private_availability_zone: ""
  private_net_1_name: contrail_net1
  private_net_1_cidr: 1.1.1.0/24
  private_net_1_pool_end: 1.1.1.254
  private_net_1_gateway: .1.1.1
  private_net_1_pool_start: 1.1.1.1

  public_net: 22d00df7-b1ca-41a1-a615-89c3bdde6c10

  template_name: test-fw-template2
  template_mode: in-network
  template_type: firewall
```

```
template_image: Pfsense
template_flavor: m1.medium
service_interface_type_list: management, right, left
shared_ip_list: False, False, False
static_routes_list: False, False, False
scaling: "True"
availability_zone: "True"
ordered_interfaces: "True"

key_name: rav_key
```

E. Enviroment file pro LbaaS - Fortigate VM

```
parameters:
  private_instance_name: test-firewall2
  max_instances: 1
  policy_name: contrail_policy2
  direction: "<>"
  start_src_ports: -1
  end_src_ports: -1
  start_dst_ports: -1
  end_dst_ports: -1

  private_availability_zone: ""
  private_net_1_name: contrail_net2
  private_net_1_cidr: 2.2.2.0/24
  private_net_1_pool_end: 2.2.2.254
  private_net_1_gateway: 2.2.2.1
  private_net_1_pool_start: 2.2.2.1

  public_net: 22d00df7-b1ca-41a1-a615-89c3bdde6c10

  template_name: test-fw-template2
  template_mode: in-network
  template_type: firewall
```

```
template_image: vsrx-15.1
template_flavor: m1.medium
service_interface_type_list: management, right, left
shared_ip_list: False, False, False
static_routes_list: False, False, False
scaling: "True"
availability_zone: "True"
ordered_interfaces: "True"

key_name: rav_key
```


Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Smola Ondřej	Polizy 16, Osice - Polizy	11475

TÉMA ČESKY:

Orchestrace a management virtuálních síťových funkcí

TÉMA ANGLICKY:

Orchestration and management of virtual network functions

VEDOUcí PRÁCE:

Ing. Vladimír Soběslav, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem této práce je analyzovat možnosti vytváření a nasazení virtuálních sítí v cloud computingu s důrazem na technologie VNF nad NFV a jejich srovnání. V rámci závěrečné práce budou analyzovány metody a nástroje pro vývoj a automatizaci služeb virtuálních sítí. V závěrečné části provede autor implementaci VNF řešení v prostředí cloud computingové platformy OpenStack.

Osnova:

1. Úvod
2. Problematika virtualizace síťových funkcí
3. Testovací prostředí
4. Příklad virtualizace síťových funkcí
5. Shrnutí
6. Závěr

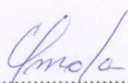
SEZNAM DOPORUČENÉ LITERATURY:

DOSTÁLEK, Libor.; KABELOVÁ, Alena. Velký průvodce protokoly TCP/IP a systémem DNS. 5. aktualizované vydání, Brno: Computer Press, a.s., 2008. 488 s. ISBN 978-80-251-2236-5.

HICKS, Michael. Optimizing Applications on Cisco Networks. 1. vydání. Indianapolis: Cisco Press, 2004. 384 s. ISBN: 978-1-58705-153-1.

HUCABY, David. CCNP SWITCH 642-813 Official Certification Guide. 1. vydání. Indianapolis: Cisco Press, 2011, 533 s. ISBN 978-1-58720-243-8.

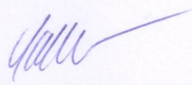
Podpis studenta:



Datum:

14. 3. 2016

Podpis vedoucího práce:



Datum:

14. 3. 2016