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 bakalářskou práci vypracoval samostatně a uvedl jsem všechny použité prameny a literaturu.

V Hradci Králové dne 13. srpna 2016 Ondřej Smola

Poděkování

Děkuji vedoucímu bakalářské 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 dynamické 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 OpenStack s SDN řešením OpenContrail.

Annotation

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

Obsah

1 Úvod

V dnešní době dochází v datových centrech k nasazování nových moderních technologii. 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 hardwarově 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 souvislé blok výpočetního výkonu a jeden souvislí blok úložiště.

Dalším takovýmto funkcionálním blokem, který je součástí datových centrech a je velice důžitou součátí infrastruktury velkých společností, jsou počítačové sítě. V oblasti počítačových sítích 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 [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. Management takovéto sítě

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 správě 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 přistupují k oblasti počítačových sítí. 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 funkcionalitu 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, jak popisují [5] a [6].

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 navržení architektury pro NFV frameworku, na kterém následně bude provedeno testovaní několika virtulálních síťových funkcí a jejich následné porovnání a zhodnocení. V celé práci budou využívány pouze aktuálně dostupně technologie, které by následně mohli být využity i produkčním prostředí.

Celá struktura této práce je rozdělena na několik části. V druhé kapitole jsou vysvětleny hlavní pojmy a problematika oblasti virtualizace síťových funkcí. Třetí se zabývá refereční architekturou NFV a popisem možných technologii. Ve čtvrté kapitole je popsána již výsledná architektura a použité technologie společné s odůvodněních 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

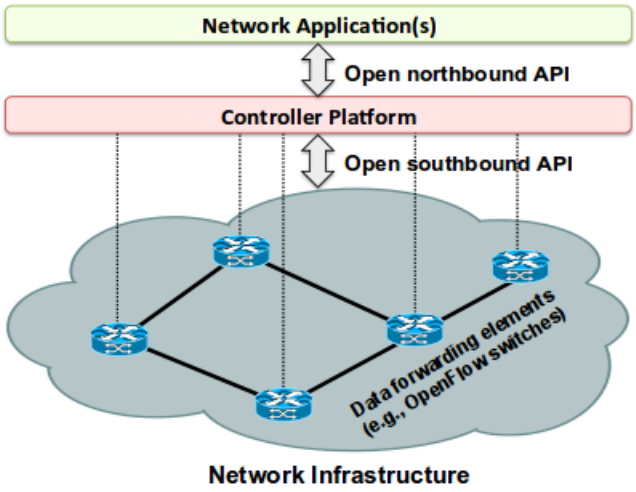
Tato kapitola se zabývá základní analýzou a popisem problematiky spojené s oblastí virtuální síťových funkcí. Nejprve uveden přehled a krátký popis virtualizace a cloud computingu spolu s jejich přínosem pro IT. Následně je vysvětlena potřeba virtualizace síťových funkcí. Zde jsou porovnány jednotlivé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

2.1 Tradiční počítačové sítě

Pohledem na tradiční počítačovou sít zjistíme, že nejvíce síťové funkčnosti je soustředěno ve fyzických proprietárních zařízeních jako jsou routery, firewally či load balancery. To znamená, že provozovatelé počítačových sítí se při spouštění nových síťových služeb musí na tyto zařízení spoléhat. Což může vést k zdlouhavému nasazování, zvýšené spotřebě energii a investici do školení pracovníků pro dané proprietární zařízení. Zároveň zde není možnost, aby síť mohla být dynamicky ovládána dle aktuálních požadavků uživatelů sítě. Například vývojář nemůže hned nasadit aplikaci do produkce. Musí nejprve čekat na síťový tým než patřičně nakonfigurují síťové prvky pro správné a bezpečné fungování celé infrastruktury.

2.2 Softwarově definované sítě - SDN

Softwarově definované sítě (SDN) je jednou z nových technologii, která se snaží vylepšit a automatizovat správu stávajících počítačových sítí. Dle [29] jde o koncept, ve které je oddělena řídící logika (control plane) z jednotlivých routerů a switchů, které přeposílají traffic (data plane). Tím, že dojde k oddělení datové a řídící vrstvy, se routery a switche stanou pouze přeposílající data a veškerá řídí 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 2.1: Schéma SDN, převzato z [29]

Obrázek č. 2.1 úkazuje jednoduché schéma softwarově definovaných sítí. 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.
- Northbound APIs - Toto API využívají aplikace pro komunikaci s SDN controllerem.
- Control vrstva - V této vrstvě je centralizována veškerá logika, které dříve byla v síťových prvcích.
- Southbound APIs - Jedná se o skupinu API protokolů, které pracují mezi vrstvou infrastruktury a control vrstvou. Jejím hlavním úkolem je komunikace, která povoluje SDN controleru instalovat na samotné síťové prvky rozhodnutí definované v aplikační vrstvě.
- 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 se navzájem se doplňují. Fakt, že SDN umožňuje programaticky ovládat počítačovou síť, lze využít pro poskytnutí programovatelné konektivity mezi jednotlivými virtuálními síťovými funkcemi. Naopak SDN může využít NFV

tím, že implementuje potřebné síťové funkce jako software. Může tak virtualizovat SDN Controller, který tak může běžet na co nejvhodnějším místě v datovém centru. Je vidět, že tyto dvě technologie se dobře doplňují, proto jsou často součástí jednoho řešení. [**?**]

2.3 Virtualizované síťové funkce - NFV

//TODO Obrazek + popis jak je to dobry //SW model

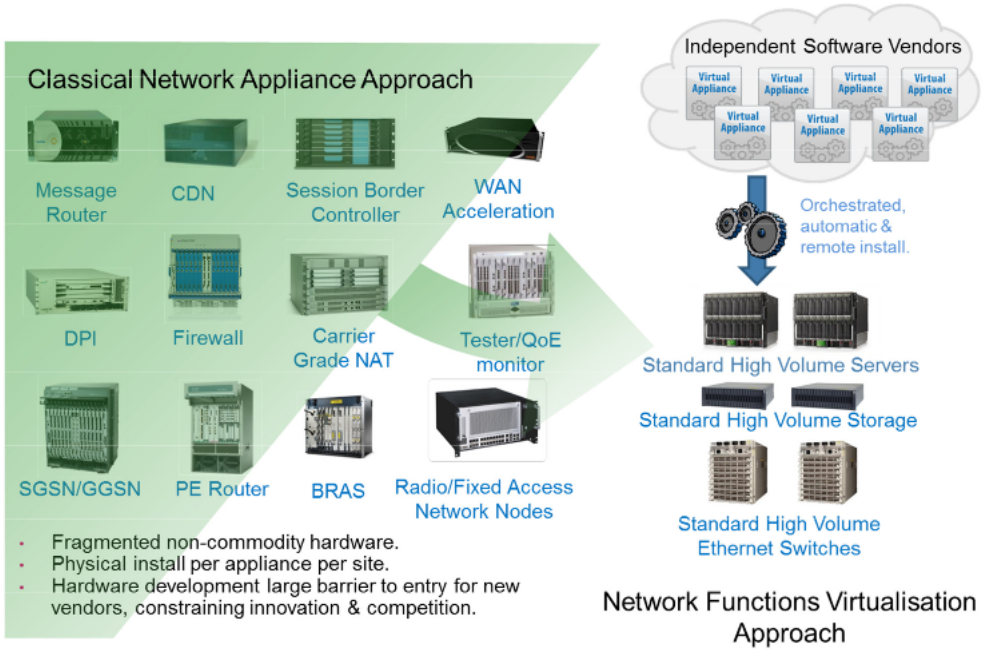
//TODO Obrazek pro Home //TODO Obrazek pro Cloud //TODO Obrazek pro Telco

Hlavní cíle tohoto řešení jsou zlepšit následující aspekty provozu telekomunikačních sítí:

- Smíření investičních nákladů – snížení potřeby nákupu jednoúčelových hardwarových zařízení, možnost platby pouze za využité 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

Jak je uvedeno v [6] a [5], tak celá myšlenka je založena na tom, že dojde k separování softwarové funkcionality v síťových prvcích od proprietárního hardwaru, na kterém běží. To umožní se síťovými funkcemi zacházek jako s klasickými softwarovými aplikacemi, které mohou běžet na standardním komerčně dostupných serverech jenž organizace v současnosti používají. Tím bude zároveň umožněno flexibilní nasazování těchto síťových funkcí a jejich dynamický provisioning. Díky tomu, že jsou síťová funkce odděleny od hardwaru, tak je také možné jejich vhodnější umístění v topologii. To znamená dle požadavků na umístění mohou být nasazeny v datových centrech, síťových uzlech či přímo v uživatelově koncovém bodě. Hlavní koncept virtualizace síťových funkcí znázorňuje obrázek č. 2.3.

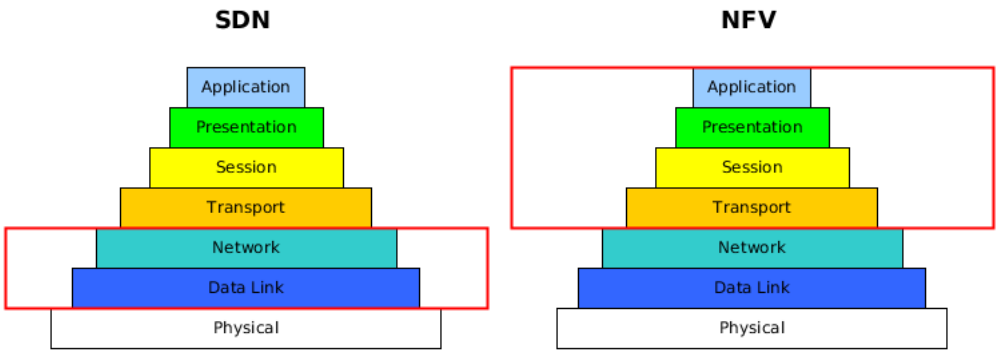
Za zmínění stojí poznámka v [6], kde je řečeno, že obecný koncept oddělení síťové funkce od hardwaru ještě nutně neznamená potřebu využití virtualizace. Protože budou síťové funkce dostupně jako software, tak mohou být nainstalovány a provozovány přímo na fyzickém stroji. Ovšem rozdíl je, že tento stroj již nebude speciální hardware,



Obrázek 2.2: Koncept virtualizace síťových funkcí (NFV)

ale klasický server. Tento scénář může být do jisté míry použit při nasazovaní 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. To vše umocňuje fakt, že většina datových center v současnosti již využívá cloud computing.

2.4 Souvislost SDN a NFV



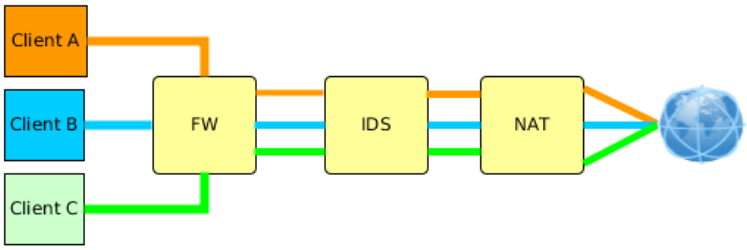
Obrázek 2.3: (NFV)

//Tabulka rozdílů porovnani
[27]

2.5 Service Chaining

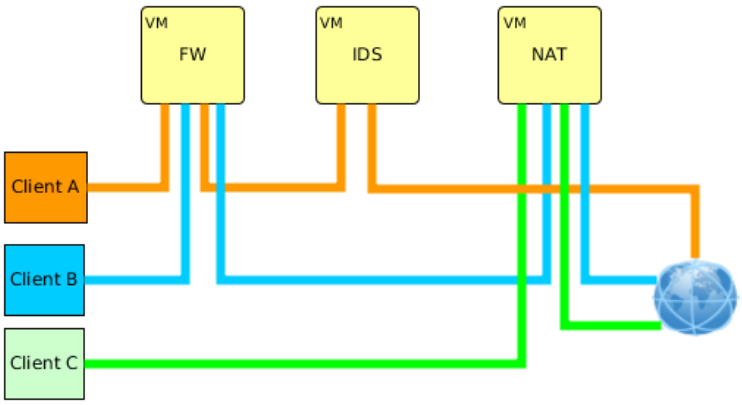
Jednou z výhod NFV je možnost využít Service Chaining. Service chaining je ve skutečnosti součást SDN. Jde o princip jakým lze dynamicky pospojovat jednotlivé VNF a ovladat tak toky v síti. [**?**]

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. 2.4. 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 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 sítě s více uživateli, kde každý požaduje jinou síťovou funkci. Potřeba jednotlivých síťových služeb se samozřejmě může v čase měnit. Příklad takové sítě lze nalézt ve většině datových center.



Obrázek 2.4: Ukázka klasického service chainigu pomocí fyzických síťových prvků

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. 2.5 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.



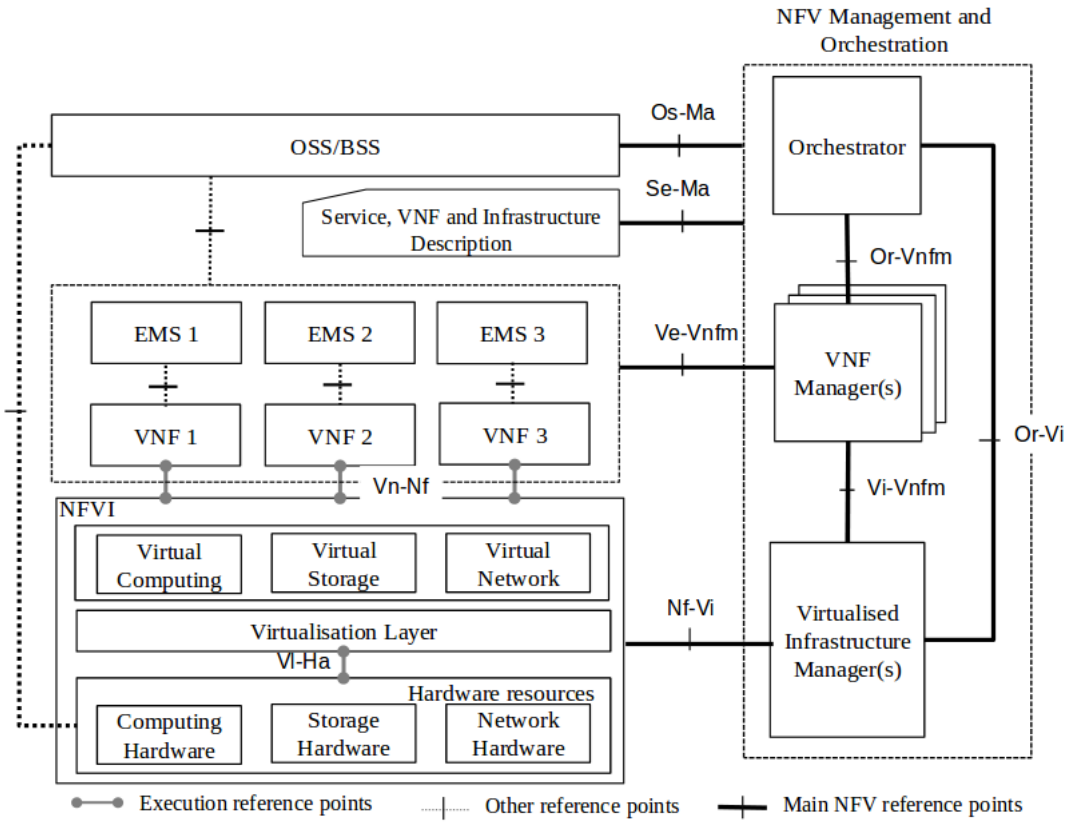
Obrázek 2.5: Ukázka VNF service chainigu

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 [7] 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.



Obrázek 3.1: NFV architektura, převzato z [7]

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 virtualizací v NFV frameworku.

Podrobné vysvětlení všech zkratek a teminologii, která je vyobrazena na obrázku lze nalézt v [8]. Pro účely této práce jsou dále popsány pouze výše uvedené hlavní části architektury.

3.1.1 Infrastruktura NFV

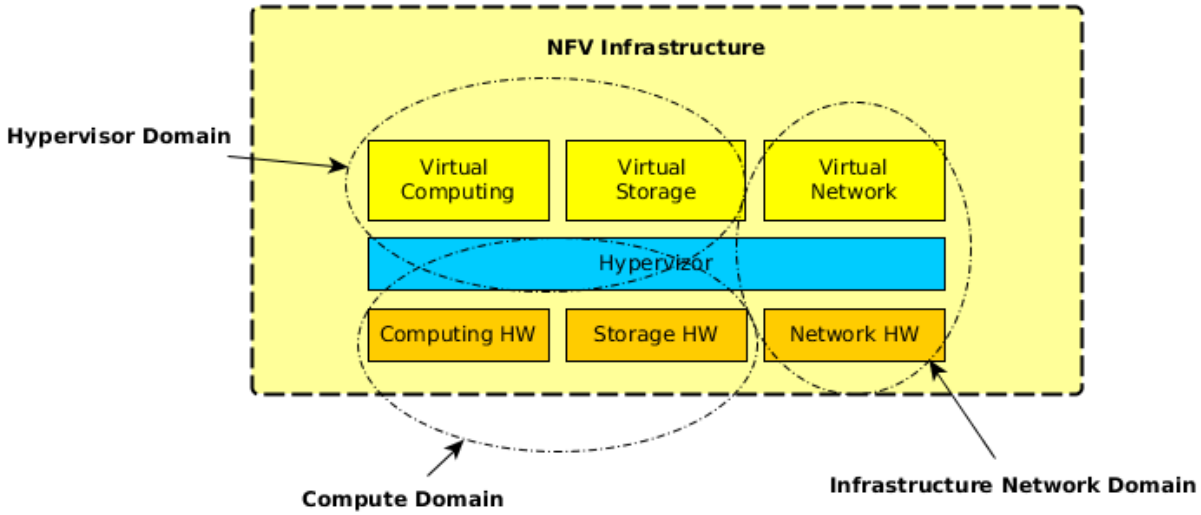
Ve zdroji [9], který detailně popisuje infrastrukturu pro virtualizaci síťových funkcí (NFVI), je uvedeno, že je v ní sdružení všech základních zdrojů potřebných pro běh virtuálních síťových funkcí (VNF). Z tohoto důvodu sem patří veškerý hardware. Do NFVI také patří některé softwarové komponenty, které 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. [10]
- 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. [11]

- Infrastructure Network Domain - V této doméně je řešeno veškeré propojení výše zmíněných domén. Tedy fyzické i virtuální infrastruktury.[12]

Funkci obsaženou v jednotlivých doménách znázorňuje obrázek č. 3.2. Více informací na tuto problematiku lze nalézt v [9] a ve zdrojích uvedených u každé domény.



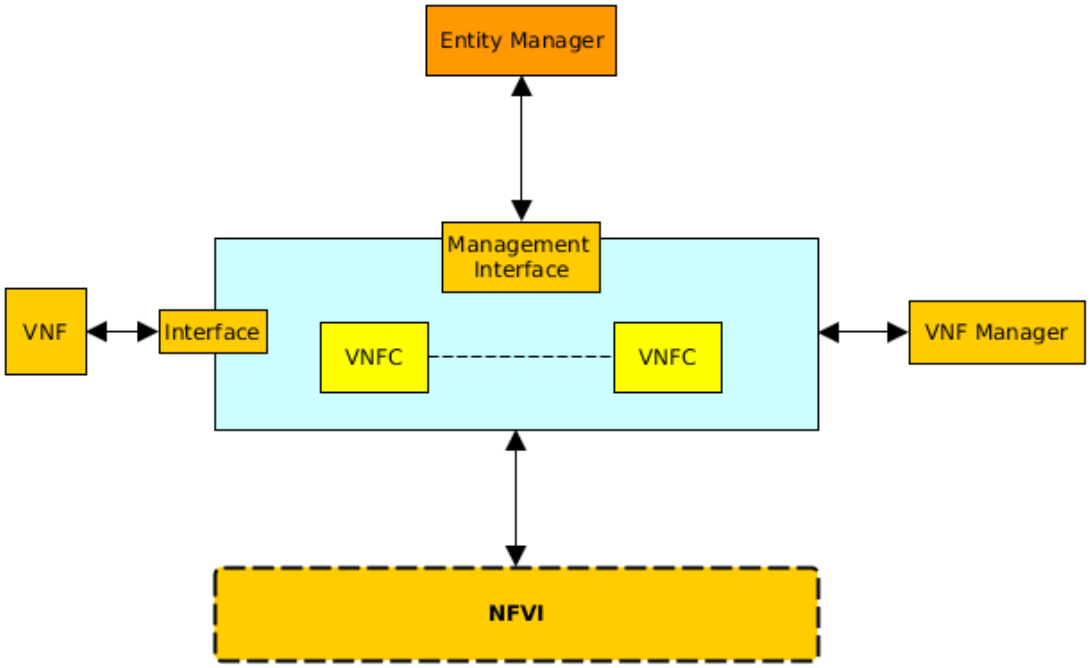
Obrázek 3.2: Schéma NFV infrastruktury

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 platforem, které lze NFVI využít.

3.1.2 Virtuální síťová funkce

Virtuální síťová funkce (VNF) je dle [13] určitá síťová funkce, která běží na NVF 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 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 [13]. Celý životní cyklus VNF (vytvoření, spuštění, zastavení, smazání a škálování) řídí VNF Manager, který je součástí NVF 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

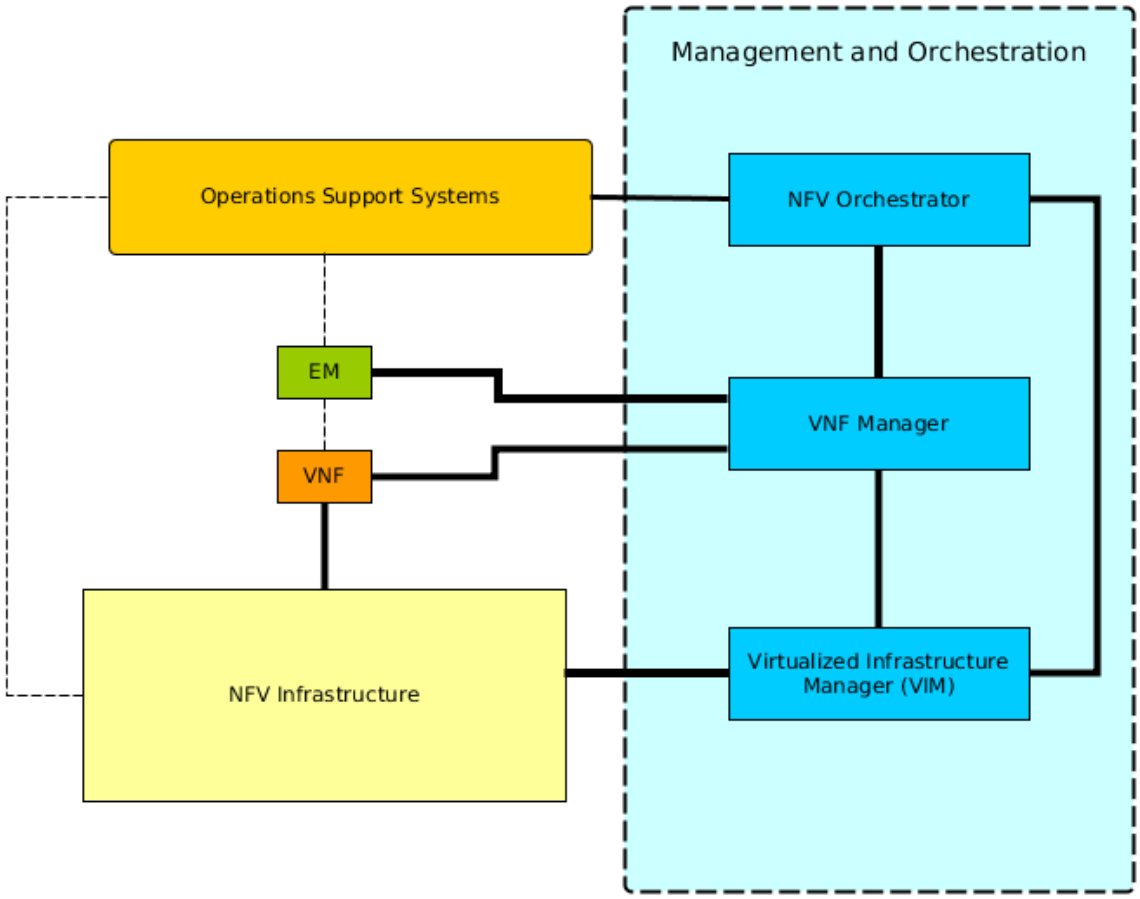
Pohledem na současný trh zjistíme, že VNF je prakticky poskytována ve 3 základních podobách.

- 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.



Obrázek 3.4: Schéma NFV MANO

Jak vyplývá z obrázku č. 3.4, tak referenční návrh MANO dle [14] se skládá ze hlavních 3 částí, které se zabývají správnou jednotlivých vrstev NFV frameworku.

- Virtualized infrastructure manager (VIM) - Řídí a spravuje fyzické a virtuální zdroje v jedné doméně infrastuktury. Celková infrastruktura se může skládat z více domén a každá musí mít svůj VIM. Jeho typickými úlohami jsou vytvýření, 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, udržuke 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 orchestator - 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 rozsáhlí vývoj a existuje několik projektů, které se tím zabývají. V článek [15] 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.

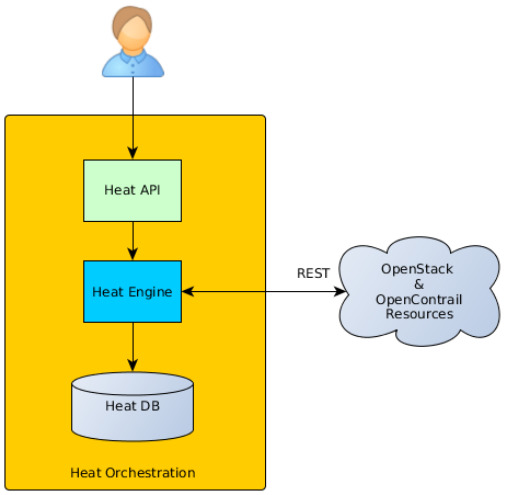
3.2.1 OpenStack

OpenStack [16] 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 webově 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 například Keystone (Autorizace a autentizace), Nova (Výpočetní služby), Horizon (Dashboard) a další. Další informace lze nalést například v [17]. 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. [18]



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 [19] , 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 vytvorena iniciativa OP-NFV [20] , 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.

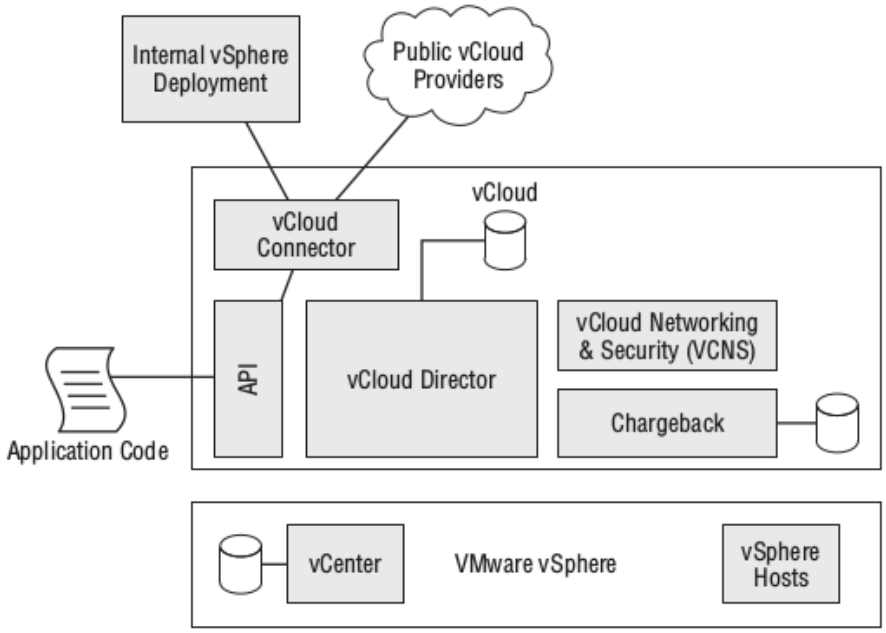
3.2.1.2 OpenContrail

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. [21]

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.

3.2.2 VMware vCloud Suite

Společnost Vmware poskytuje pro privátní cloudové systémy své řešení, které označuje jako VMware vCloud. Toto řešení je poskládané z jednotlivých produktů této společnosti. Celá architektura má hierarchická model, který je vidět na obrázku č. 3.6



Obrázek 3.6: Schéma VMware vCloud, převzato z [31]

Dle [31] se VMware vCloud skládá především z těchto komponent.

- VMware vCloud Director - je jedna ze základních součástí potřebných pro vytvo-

ření 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) - hypervisor, který byl popsán v kapitole **??**.
  - VMware vCenter Server - umožňuje efektivní a pokročilejčí 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 vituální prostředí. Poskytuje mnoho síťových funkcí a poskytuje framework pro integraci řešeí 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.

3.2.2.1 VMware NSX

3.3 Dostupné možnosti pro VNF

Navrhnutá řešení v této práci předvádějí virtuální víťové funkce pro firewall a load balancing.

3.3.1 FWaaS

Pokud se podíváme na trh s VNF u některých vendorů, tak zjistíme, že mnozí poskytují virtuální instance, které se dají použít pro účely VNF v této práci. Tato práce je zaměřena především na funkce firewallu a proto zde jsou uvedeny příklady pouze pro ně. Uvedeny jsou hlavně produkty největších a nejpoužívanějších poskytovatelů síťových prvků a také open-source firewall.

- 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 bezpečností funkce pro poskytovale telekomunikačních služeb a větší společnosti. Toto VM je určené pro privátní, public i hybrid cloud.
- 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.
- Cisco ASAv - Společnost Cisco nabízí Adaptive Security Virtual Appliance (ASAv), která obsahuje stejný software jako fyzické ASA zařízení a vetšinu funkcí pro firewall, routing a VPN.
- PFSense - PFSense je open-source projekt, který má za cíl poskytnout firewall postavený na operačním systému FreeBSD, který může běžet na klasické architektuře jednodeskových počítaču. Toto řešení poskytuje všechny důležité vlastnosti komerčních firewallů, má jednoduché ovládání a je to otevřené řešení.

3.3.2 LbaaS

- AVI networks
- HAproxy – je velmi rychlé a spolehlivé řešení nabízející vysokou dostupnost, load balancing a proxy pro aplikace založené na TCP a HTTP

3.4 Možnosti prostředky pro Management a Orchestraci VNF

Přestože

Vzhledem k tomu, že v této práci je uvažováno využití cloudové platformy, tak je možné celé MANO zjednodušit na management VNF. Je to z důvodu toho, že cloudová platforma dokáže sama řídit celý žitovní cyclus VNF

//*Konfigurovační management //***Co existuje //***SaltStack //***Ansible

//*Predkonfigurovaneho image či aplikace //*Pomoci Heat bash scriptu

4 Požadavky a architektura prostředí pro NVF 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. Tato kapitola bude již věnována konkrétnímu příkladu využití virtuálních síťových funkcí v cloudovém prostředí. Nejprve zde popsána navržená architektura pro privátní cloudovou platformu využívající virtualizaci síťových funkcí, kterou mohou využívat všichni její uživatelé. Pro tuto cloudovou platformu a pro její uživatele byli navrženy dva příklady virtuálních síťových funkcí. U obou příkladů jsou uvedeny scénáře a způsob jakým jsou navrženy.

4.1 Požadavky na NFV Infrastrukturu

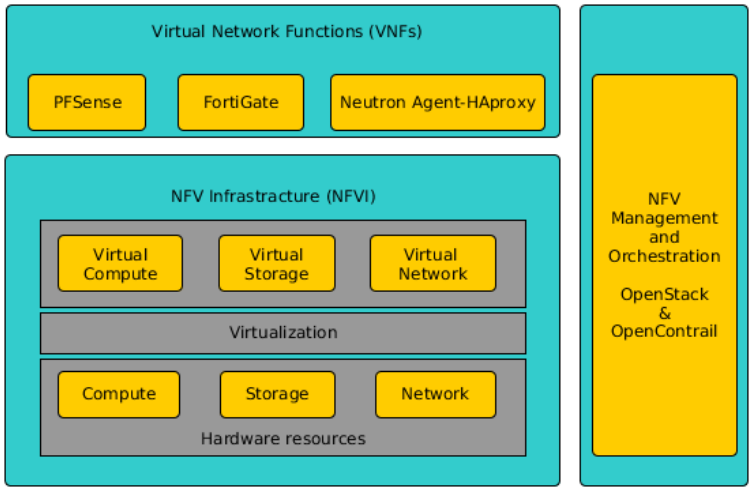
//Open Source technologie
//Možnost automatického vytváření ifrastruktury
//Podporara různých řešení pro SDN
//Vybral jsem si OpenStack protože ...

4.2 Požadavky na VNF

Vybral jsem si PFSense a Fortigate ...
Vybral jsem si HAproxy a AVI networks ...

4.3 Výsledná architektura použitého frameworku

Architektura navrženého řešení byla implementována pomocí cloudové platformy OpenStack a SDN řešení OpenContrail. Obrázku č. 4.1 znázorňuje tyto technologie v souvislosti s referenční architekturou popsanou v kapitole **??**. Je nutné říci, že obě technologie nezapadají přímo do jedné z částí referenční architektury. Naopak v některých případech se překrývají nebo se v ní doplňují.



Obrázek 4.1: Architektura NFV řešení

OpenStack byl zvolen, protože se jedná o největší open-source cloudovou platformu na světě. OpenStack tvoří část správy infrastruktury. Hardwarová vrstva infrastruktury se může skládat z libovolných serverů, na kterých je nainstalován KVM hypervizor. Tento hypervizor tvoří virtuální vrsvu a byl vybrán, protože je nejčastěji používán společně s OpenStackem. Avšak v případě potřeby by zde mohl být použit i jiný hypervizor, pokud bude zachována kompatibilita vůči OpenStacku.

OpenStack spravuje převážně zdroje týkající se výpočeního výkonu (Compute) a uložiště (Storage). Tyto zdroje následně přiděluje dle potřeby virtuálním instancím nebo v našem případě instancím, které slouží jako VNF. Bylo však nutné zvolit řešení, které se bude starat o síťování.

Speciálně pro vyřešení síťování v této infrastruktuře je součástí řešení OpenContrail. Díky tomu je možné vytvářet overlay sítě pomocí VXLAN či MPLS over GRE, kterými jsou dynamicky propojovány jednotlivé VM a VNF.

Jednotlivá VNF mohou být v OpenContrailu vytvořena pomocí tzv. Servisních Instance a Servisní Templatů. Ty budou v této práci použity pro vytvoření VNF sloužící jako firewally a budou podrobně popsány v kapitole věnující se vytváření této služby.

Další součástí, která musela být v architektuře navrhnuta, je způsob řízení a správy jednotlivých VNF. Zde se muselo jednat o řešení, jakým automaticky vytvořit a popřípadě i smazat všechny potřebné části potřebné pro VNF. Pro tuto část byl zvolen Heat. Heat je část OpenStacku, která slouží pro automatickou orchestraci. Ten bude v tomto návrhu zastávat roli VFN managera, pomocí kterého budou jednotlivé VNF spravovány. Avšak dalo by se říci, že do této role spadá i OpenContrail, protože právě on umožnuje také spravovat jednostlivá VNF za běhu.

Heat je hlavní projekt v OpenStacku pro orchestraci. Umožňuje uživatelům popsat nasazení komplexních cloudových aplikací v jednom textovém souboru, který se nazývá Heat template. Tyto soubory se dají předat heat enginu, který podle nich dokáže automaticky vytvořit požadované zdroje v OpenStacku i v OpenContrailu.

Z toho návrhu je patrné, že zde není implementovaný NFV orchestrator. Je to z důvodu toho, že pro účely řešení virtuálních síťových funkcí na cloudové platformě OpenStack s OpenContrailem, která je navržena v této práci, není tato část potřeba.

5 Testovací scénáře a realizace pro VNF a NFV

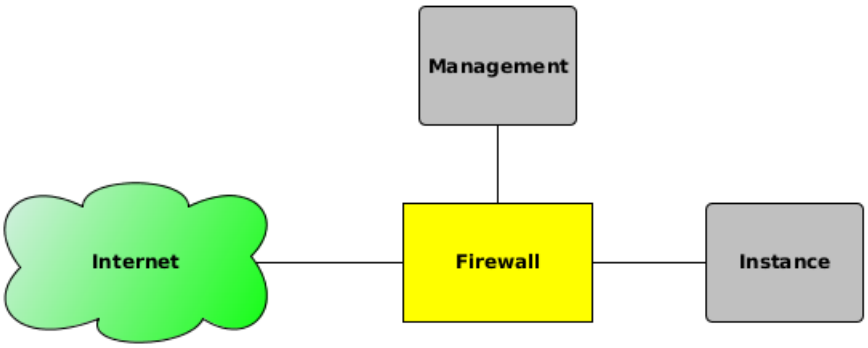
V předchozí kapitole byla popsána oblast virtualizace síťových funkcí a jeji architektura. Také byly popsány jednotlivé technologie, které budou v této kapitole použity k realizaci ukázkových VNF. Pro každou VNF zde bude uveden příklad jejího použití a jakým způsobem jsou realizovány požadavky na její životní cyclus, které byly uvedeny v předchozí kapitole.

5.1 Scenáře pro použití vybraných VNF

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 by následně použil fyzický load balancer. Ten bude spravovat příchozí komunikaci a distribuovat ji mezi několika serverů. Tím bude zajištěna rozloha zátěže a zajištěn bezvýpadkový provoz. Nevýhodou toho přístupu je právě nutnost pořízení fyzického load balanceru. Tím se uživatel značně omezí ve flexibilitě. Pokud například bude chtít další webové služby, které by měli být oddělené od těch stavajících, tak si bude muset opět pořizovat další hardwarový prvek. Alternativou k tomuto přístupu je využití cloudu a VNF, která bude mít load balancing funkcionalitu.

- Webové servery - Virtuální instance, na kterých bude umístěna požadovaná webová aplikace.
- Privátní síť - Je síť, kde budou tyto servery umístěny.
- Load balancer - Tato část je zodpovědná za řízení příchozí a odchozí komunikace webových serverů s okolním světem (Internetem).



Obrázek 5.1: Firewall as a Service

5.1.2 Scénář FwaaS

5.2 Realizace VNF pro LbaaS

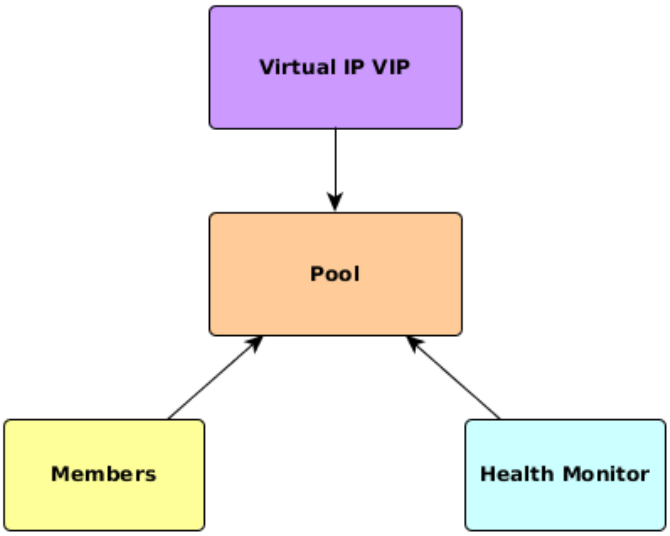
5.2.1 HAproxy - Neutron HAproxy agent

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.

V této práci je ukázán příklad využití implementace load balanceru v OpenContralu, který může být přes toto api ovládán. Tento příklad je však univerzální a může být použit s jakoukoli implementací load balanceru, at už virtuláního (sofwarového) či fyzického, pokud dokáže komunikovat s OpenStack Neutron LbaaS rozhraním. Dle dokumentace [**?**] je v OpenContrailu implementace load balanceru řešena pomocí HAProxy. HAProxy je zdarma dostupný open source software pro unix operační systémy [34].

Load balancer se v Neutron LbaaS skládá ze 4 objektů.

- Pool - Označuje síťový rozsah pro webové servery.
- Virtuální IP (VIP) - ip adresa, na kterou přichází komunikace
- Member - Označuje konkrétní instanci, která je členem poolu.
- Monitor - Monitoruje stav jednotlivých serverů a aplikací.



Obrázek 5.2: Neutron LbaaS

Obrázek č. 5.2 zachycuje jednotlivé závislosti mezi těmito objekty. Tato implementace má tyto hlavní funkce:

- Poskytuje Load balancing komunikace od klientů do poolu serverů. Load balancer zprostředkovává spojení prostřednictvím své VIP.
- Poskytuje load balancing pro HTTP, TCP a HTTPS komunikaci.
- Poskytuje možnosti pro monitorování aplikací. Prostřednictvím HTTP, TCP či PING. Zde celý proces tak, že se load balancer pokusí v určeném časové intervalu navázat s daným serverem v pool spojení dle vybraného protokolu.
- Umožňuje asociaci floating ip (veřejné adresy) k VIP, čímž umožňuje přístup k serverů z veřejné sítě.

Celý proces probíhá tak, že každý virtuální server, který je asociovaný s daným poolem z něj obdrží IP adresu. Když příjde 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. Výběr serveru může probíhat pomocí jedné z následujících metod:

- Round robin - zde se komunikace distribuuje rovnoměrně, resp. dle vah zadaných u jednotlivých memberů v poolu.

- Least connection - zde je vybran member s nejméně spojeními.
- Source IP - u této metody je vybrán member na základě zdrojové ip adresy klienta.

V tomto případě si tedy není třeba starat o automatizaci konfigurace celého řešení. Je zde pouze nutné správně nadefinovat jednotlivé komponenty v heat templatu, abychom dosáhli požadovaného chování.

5.2.1.1 LbaaS heat template

Aby nemusel uživatel ručně vytvářet load balancer ručně, tak byl celý proces vytváření load balanceru zautomatizován pomocí heat templatu. Navržený heat template pro LbaaS v sobě obsahuje několik prostředků, které se po jeho spuštění pokusí heat engine vytvořit. Celý template v sobě obsahuje i webové instance, které slouží pro testování. V produkci by však byly v odděleném templatu. Template je parametrizovaný a konktétní hodnoty pro jednotlivě zdroje (ip adresy, ip) jsou v tzv. enviroment file, který se zadává při spouštění daného templatu. Dále jsou popsány pouze hlavní části heat templatu.

- privatni síť - k této síti jsou připojeny obě webové instance, load balancer a router. Součástí je definice toho zdroje jsou je i subnet, který má dále paramentry týkající se DHCP ip adres.

```
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 }
```

Ukázka kódu 5.1: Privátní síť a subnet

- 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.

```
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
```

Ukázka kódu 5.2: Web server 1

- router - toto je Neutron router implementují SNAT. V tomto příkladě je využívaný webovými servery pro konektivitu k Internetu. Toto je nutné pro nainstalování programu Apache na webové servery.

```
router:
  type: OS::Neutron::Router
  properties:
    name: { get_param: router_name }
    external_gateway_info:
      network: { get_param: public_net_id }
```

Ukázka kódu 5.3: Web server 1

- public síť - toto je veřejná síť, ze které je získána VIP pro load balancer. Na tuto VIP bude dále asociována floating ip.

```
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 }
lb_floating:
  type: OS::Neutron::FloatingIP
  properties:
    floating_network_id: {get_param: public_net_id}
    port_id: {get_attr: [lb_pool, vip, port_id]}
```

Ukázka kódu 5.4: Public síť a subnet

- 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 nejjednodušímu 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
      address: { get_param: public_net_ip }
      admin_state_up: True
      subnet: { get_resource: public_subnet }
```

Ukázka kódu 5.5: Load balancer pool

- members - po vytvoření instancí je nutné jejich přidání do poolu jako membery. 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
```

Ukázka kódu 5.6: Members

- 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 5.7: Monitor

V celém heat templatu je ještě více zdrojů, které se vytváří. Ty zde však nebudou popsány. V případě zájmu lze nálest komletní heat template v příloze.

5.2.1.2 Testování LbaaS

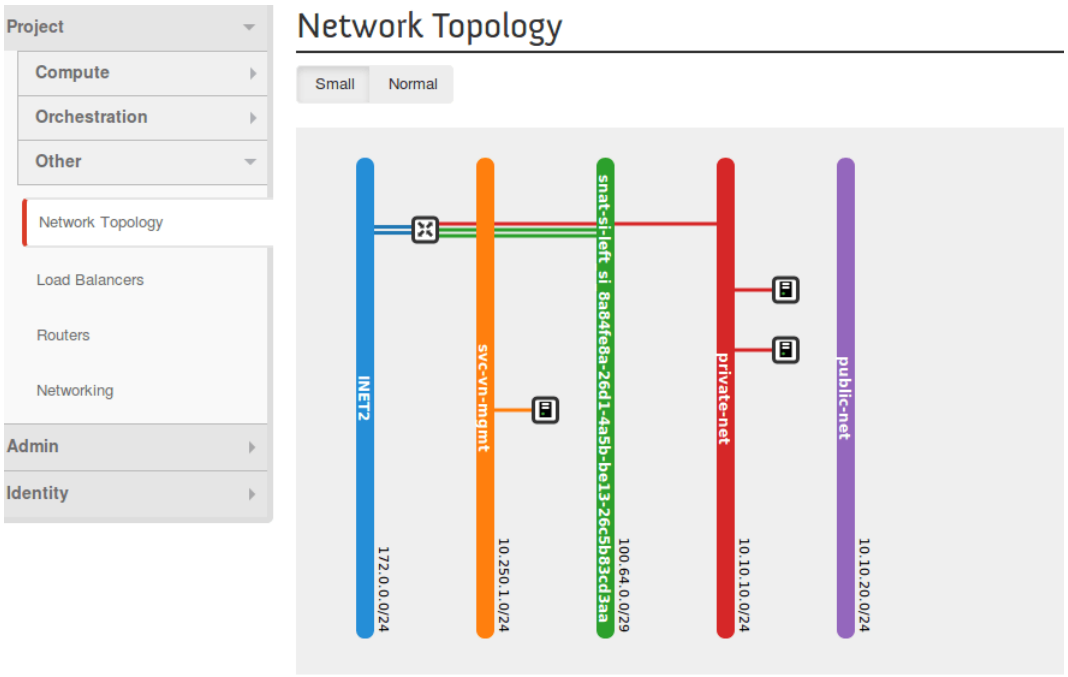
V reálné případě by si uživatel heat tempalte vybral z katalogu. Avšak v případě této práce bude heat template spouštěn pomocí příkazu v terminálu:

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

Tento příkaz vytvoří všechny již uvedené prostředky pro load balancing. Konkrétní load balancer má nakonfigurovanou virtual ip adresu (VIP) a k ní přiřazenou floating adresu, která je přístupná z externích sítí. Zároveň má tento load balancer přiřazený

pool, ke kterému je přiřazena přiřazena privátní síť 10.10.10.0/24. Další zdrojem, který byl vytvořen je health monitor. Díky němu má load balancer přehled o aktuálním stavu webových instancí. Pokud by náhodou některá z nich přestala odpovídat, v tomto případě na ping, tak by load balancer na tuto instanci přestal zasílat traffic.

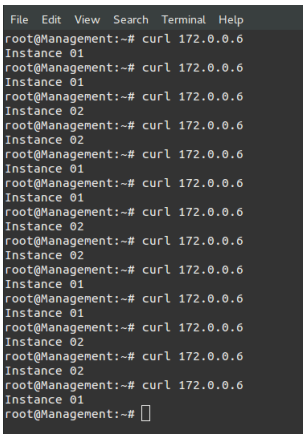
Na obrázku č. 5.3 je zobrazek screenshot vytvořené topologie v OpenStack dashboardu. Jsou zde vidět vytvořené servery a sítě. Není zde zobrazen load balancer, protože tato vizualizace tento prvek nezobrazuje. Lze ho nalézt v jiné části dashboardu, ale pro názornost bude rovnou otestováno jeho správné chování.



Obrázek 5.3: Vytvořená síťová topologie

Otestování správného chování virtuálního load balanceru, lze provést opakovaným dotazem na právě vytvořené webové servery. Tím je bude zároveň otestována jejich správná konfigurace. Pokud by totiž nevrátili správnou odpoveď 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. 5.4. 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.



Obrázek 5.4: Test konektivity a load balancingu

5.2.2 AVI networks

5.3 Realizace VNF pro FwaaS

5.3.1 Servisní instance v OpenContrailu

V OpenContrailu je sice možnost využívat implementaci routeru s SNAT, která umožnuje 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.

Servisní instance v OpenContrailu je jednoduše virtuální stroj, který poskytuje danou VNF. Úplně nejjednodušším příkladem může být virtuální stroj s operačním systémem GNU/Linux, který může sloužit jako router mezi dvěma sítěmi. Pro vytvoření takového virtuálního stroje jsou nutné 3 základní elementy.

- Service Template
- Service Instance
- Service Policy

Servisní Template obsahuje obecný předpis pro danou VNF v OpenContrailu. Pro správné fungování je nutné zadat nastavit správné parametry patří:

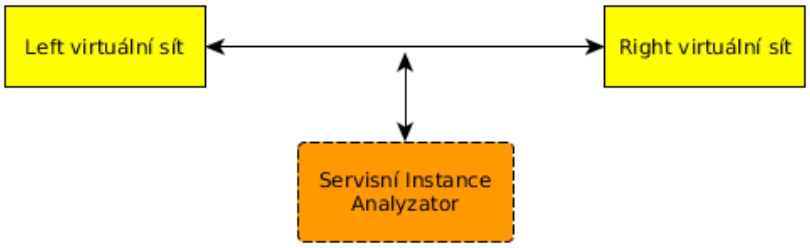
- Název - Název je označení daného Servisního Templatu. Pomocí něho lze následně identifikovat daný template a spustit dle jeho parametrů Servisní instanci.

- 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 navracející 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.

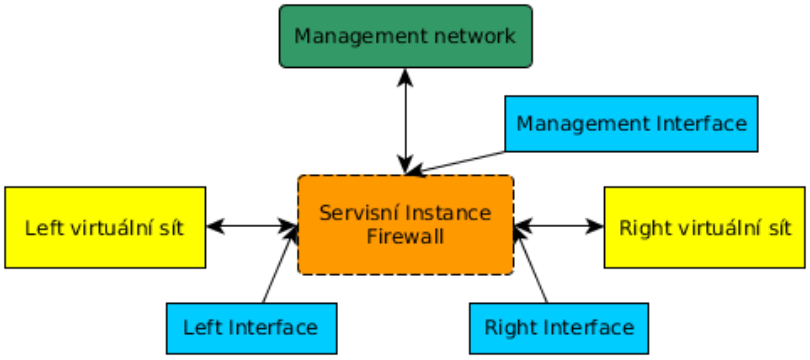
První z nich je Analyzer. Ten slouží k analýze a zachytávání síťového trafficu. Image pro tento typ servisní instance obvykle obsahuje protokolový analyzér a paketový sniffer, jako je například oblíbený program Wireshark. Tato instance dostává traffic, který je posílán mezi dvěma sítěmi. Tento traffic 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 čí jiným entitám v síti. Obrázek č. 5.5 znázorňuje tento typ servisní instance.

Druhým typen servisní instance 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 třetí síťový port, který slouží pro out-of-band management. Přestože některá řešení pro servisní instance sloužící jako firewall mohou mít již své požadované chování definované hned při jejich



Obrázek 5.5: Schéma zapojení servisní instance Analyzer

startu, tak tento port může být velice užitečný při konfiguraci dané servisní instance. A to ať už se jedná o konfiguraci manuální či pomocí nějaké vyšší management entity.



Obrázek 5.6: Schéma zapojení servisní instance

Service policy dovoluje síťový traffic mezi virtuálními sítěmi a říká systému, aby ho posílal skrze servisní instanci.

5.3.2 Heat template pro FwaaS

Pro FwaaS je narhnut heat template, který obsahuje:

- privátní síť

```
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 }
```

Ukázka kódu 5.8: Privátní síť

- firewall template

```
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 ] }
```

Ukázka kódu 5.9: Firewall servisní instance

- firewall instance

```
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}
            }
        ]
```

Ukázka kódu 5.10: Privátní síť

- virtuální instance

```
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
```

Ukázka kódu 5.11: Virtuální instance pro testování

- contrail policy

```
 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}}]
      },
]
```

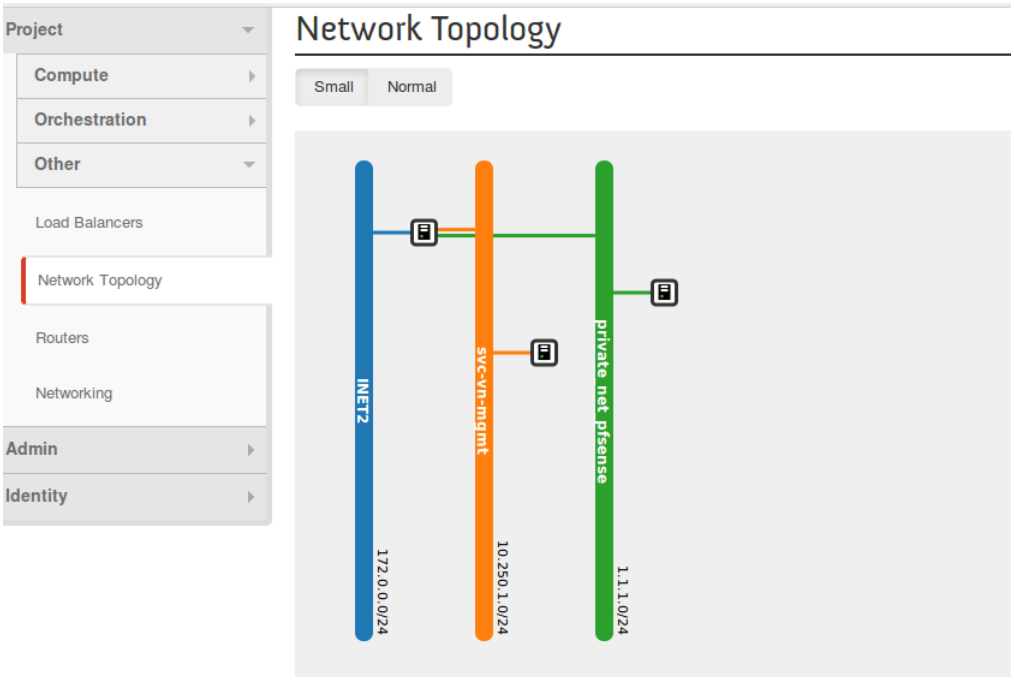
Ukázka kódu 5.12: Contrail network policy

5.3.3 PfSense

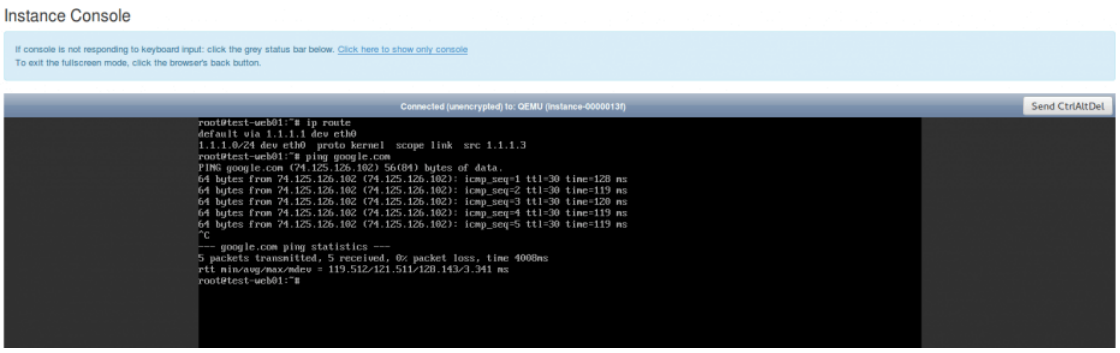
Pro vytvoření heat stacku s PFSense z templatu lze použít příkaz:
`heat stack-create -f heat/templates/fwaas_mnmg_template.hot -e heat/env/fwaas_pfsense_env.env pfsense`
a pro vytvoření heat stacku s Fortigate VM jde vytvořit pomocí příkazu:
`heat stack-create -f heat/templates/fwaas_mnmg_template.hot -e heat/env/fwaas_fortios_contrail.env fortios`

By default, pfsense firewall is configured to NAT after the heat stack is started. As a result, there is no need to make any configuration for this function. Pfsense image was preconfigured with DHCP services on every interface and there is outbound policy for NAT.

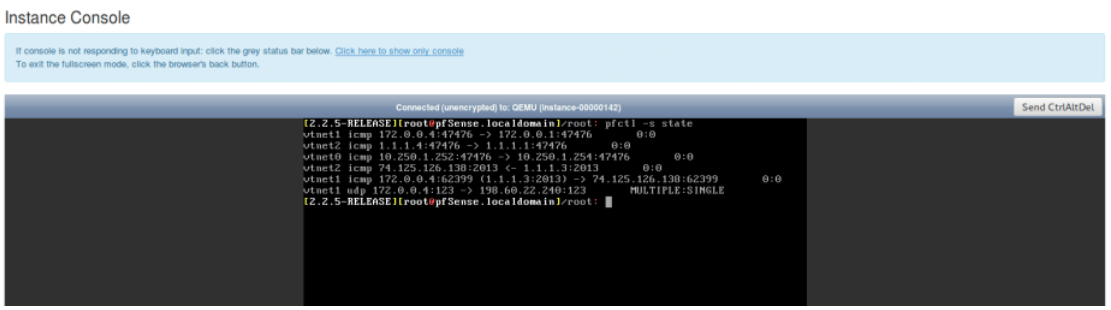
After we start the heat with pfsense there is already functional service chaining. Testing instance has default gateway to contrail and contrail redirects it to pfsense.



Obrázek 5.7: Síťová topologie



Obrázek 5.8: Test konektivity PFSense



Obrázek 5.9: Ukázka NAT session

//Bash script nefunguje //Predpripraveny image

5.3.4 Fortigate

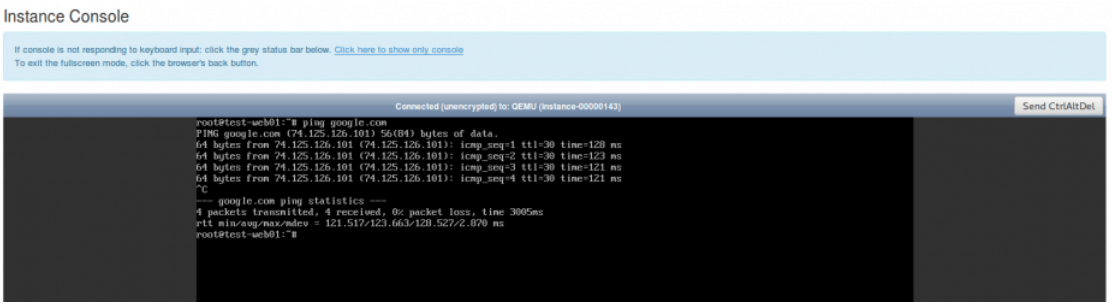
//Python API = Script // => Salt module

```
root@mnmg01:~# python fortios_intf.py
This is the diff of the conigs:



This is how to reach the desired state:
    config system interface
        edit port1
          set allowaccess ssh ping http https
        next
        edit port2
          set defaultgw enable
        next
        edit port4
          set mode static
        next
        edit port5
          set mode static
        next
        edit port6
          set mode static
        next
        edit port7
          set mode static
        next
        edit ssl.root
          set mode static
        next
    end

root@mnmg01:~#
```

Obrázek 5.10: Fortigate VM intergace konfigurace

```
ubuntu@Management:~$ ssh root@172.0.0.5
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.19.0-26-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Jan 12 10:03:49 2016 from mgmtserver14041vag
root@mnmg01:~# ls
fabfile.py   fortigate-formula  fortios_intf.txt  fortios_nat.py   param.py  update.sh
fabfile.pyc  fortios_intf.py    fortios_nat.conf  fortios_nat.txt  text.py
root@mnmg01:~# python fortios_nat.py
This is the diff of the conigs:



This is how to reach the desired state:
    config firewall policy
        edit 1
          set nat enable
          set service ALL
          set schedule always
          set srcaddr all
          set dstintf port2
          set srcintf port3
          set action accept
          set dstaddr all
          set logtraffic all
        next
    end

root@mnmg01:~#
```

Obrázek 5.11: Fortigate VM NAT konfigurace



Obrázek 5.12: Test konektivity

6 Závěr

Je v paráda.

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 GORRICHO, 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] 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

[8] 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]. ttp://www.etsi.org/deliver/etsigs/NFV/001099/003/01.02.0160/gsNFV003v010201p.pdf

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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

[15] MIJUMBI, Rashid, Joan SERRAT, Juan-luis GORRICHO, 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

[16] *Openstack: Open source cloud computing software*, 2016. openstack.org [Online]. OpenStack Foundation, 2016 [cit. 2016-08-13] Dostupné z: https://www.openstack.org

[17] 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.

[18] *Heat; OpenStack Orchestration* openstack.org [online]. OpenStack Foundation, 2016 [cit. 2016-08-13] Dostupné z: https://wiki.openstack.org/wiki/Heat

[19] *OpenStack User Survey*. In: openstack.org [online]. OpenStack Foundation, 2016 [cit. 2016-08-13]. Dostupné z: https://www.openstack.org/assets/survey/April-2016-User-Survey-Report.pdf

[20]

[21] RIJSMAN, Bruno a Ankur SINGLA. *Day One: Understanding OpenContrail Architecture*. Juniper Networks Books, 2013

[22] http://www.itproportal.com/2015/08/21/the-top-enterprise-firewalls-of-2015/

[23] KUSNETZKY, Dan. *Virtualization: a manager's guide*. Sebastopol, CA: O'Reilly, c2011. ISBN 1449306454.

[24] FONSECA, Nelson L. S. da. a Raouf BOUTABA. *Cloud services, networking, and management*. Hoboken, New Jersey: Wiley, 2015. ISBN 9781118845943.

[25] 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.

[26] 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

[27] 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

[28] NANDUGUDI, Anandatirtha, Massimo GALLO, Diego PERINO a Fabio PIANESE. Network function virtualization: through the looking-glass. Annals of Telecommunications [online]. , - [cit. 2016-08-13]. DOI: 10.1007/s12243-016-0540-9. ISSN 0003-4347. Dostupné z: http://link.springer.com/10.1007/s12243-016-0540-9

[29] KREUTZ, Diego, Fernando M. V. RAMOS, Paulo ESTEVES VERISSIMO, Christian ESTEVE ROTHENBERG, Siamak AZODOLMOLKY a Steve UHLIG. *Software-Defined Networking: A Comprehensive Survey*. Proceedings of the IEEE [online]. 2015, 103(1), 14-76 [cit. 2016-04-09]. DOI: 10.1109/JPROC.2014.2371999. ISSN 0018-9219. Dostupné z: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6994333`

[30]

[31] GALLAGHER, Simon a Aidan DALGLEISH. *VMware private cloud computing with vCloud Director*. Indianapolis, Ind.: Sybex, c2013

[32]

[33]

[34]

[35]

Přílohy

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Univerzita Hradec Králové
Faculty of Informatics and Management
Akademický rok: 2015/2016

Studijní program: Applied Informatics
Forma: Full-time
Obor/komb.: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Smola Ondřej	Polizy 16, Osice - Polizy	I1475

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 vytvářeni a nasazeni 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:** .....

Podpis vedoucího práce: ..... **Datum:** .....