Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Analýza nástrojů pro síťovou virtualizaci

Bc. Richard Felkl

Diplomová práce

2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury. Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně

V Pardubicích dne 17. května 2017 Bc. Richard Felkl

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Richard Felkl**
Osobní číslo:	**I15201**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Analýza nástrojů pro síťovou virtualizaci**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je představit, analyzovat a porovnat možnosti a přístupy k síťové virtualizaci. Autor práce v teoretické části na základě rešerše představí aktuální přístupy k řešení síťové virtualizace a to jak interní, tak externí síťové virtualizace. Následně autor představí vhodné nástroje pro síťovou virtualizaci a provede jejich komparativní analýzu vycházející z jejich systémových požadavků a využitelnosti.
V praktické části autor navrhne využití vybraných virtualizačních nástrojů pro síťovou infrastrukturu a představí jejich praktické využití včetně konfigurací a práce s vybranými nástroji.

Rozsah grafických prací:

Rozsah pracovní zprávy: **80**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

CHAYAPATHI, Rajendra. Network function virtualization (nfv) with a touch of sdn. 1st edition. Indianapolis, IN: Addison-Wesley Professional, 2016. ISBN 9780134463056.

PUJOLLE, Guy. Software Networks : Virtualization, SDN, 5G and Security. 1. London, United Kingdom: ISTE Ltd and John Wiley & Sons Inc, 2016. ISBN 9781848216945

NADEAU, Thomas a Ken GRAY. Network Function Virtualization : Service Function Chaining. 1. San Francisco, United States: ELSEVIER SCIENCE & TECHNOLOGY, 2016. ISBN 9780128021194.

Vedoucí diplomové práce: **Mgr. Josef Horálek, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2016**

Termín odevzdání diplomové práce: **17. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2016

PODĚKOVÁNÍ

Rád bych tímto poděkoval vedoucímu mé diplomové práce Mgr. Josefu Janu Horálkovi, Ph.D za cenné rady a názory při její tvorbě. Dále můj vděk patří Jakubu Pavlíkovi za pomoc s výběrem obsahu této práce a celému týmu Mirantis Czech za pomoc při získávání zkušeností nutných pro vypracování této práce. Také děkuji Marcele Zigové za pomoc s formální úpravou. V neposlední řadě bych chtěl poděkovat svým rodičům a blízkým za podporu při mém studiu.

ANOTACE

Tato diplomová práce se zabývá analýzou nástrojů pro virtualizaci prostředků v počítačových sítích. Jsou zde představeny nástroje pro testování a výuku, framework NFV, jeho komponenty, příklady užití v produkčních prostředích a porovnání možných konkrétních implementací. Praktický příklad popisuje postup při tvorbě předpisu pro automatizaci nasazení virtuální síťové služby.

KLÍČOVÁ SLOVA

virtualizace, kontejnery, počítačová síť, framework NFV

TITLE

Analysis of Computer Network Virtualization Techniques

ANNOTATION

This thesis focuses on the analysis of tools for virtualization of network devices. Firstly, the learning and testing tools are introduced. Next, the NFV framework and its components are described. Additionally, the examples of its production use are illustrated, and possible specific implementations are compared. The analytical part focuses on the process of creating the formula for automatization of virtual network function deployment.

KEYWORDS

virtualization, containers, computer networks, NFV framework

OBSAH

SEZNAM ILUSTRACÍ

SEZNAM ZKRATEK A ZNAČEK

AMD	Advanced Micro Devices
API	Application Programming Interface
ASA	Adaptive Secirity Appliance
AWS	Amazon Web Services
BSS	Business Support System
CLI	Command Line Interface
CSR	Cloud Services Router
DPDK	Data Plane Development Kit
EMS	Element Management System
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HW	Hardware
IaaS	Infrastructure as a Service
IPC	Inter Process Communication
IPFIX	IP Flow Information Export
KVM	Kernel Based Virtual Machine
LACP	Link Aggregation Control Protocol
LXC	Linux Containers
MANO	Management and Orchestration
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
NFVO	Network Functions Virtualization Orchestrator
NIS	Network Information Service
OS	Operační systém
OSPF	Open Shortest Path First
OSS	Operations Support System
PaaS	Platform as a Service
PCIe	Peripheral Component Interconnect Express
PF	Physical Function
PID	Process ID
PoP	Point of Presence
QEMU	Quick Emularor

REST	Representational State Transfer
RSPAN	Remote Switch port Analyzer
SaaS	Software as a Service
SDN	Software Defined Network
SR/IOV	Ringle Root Input/Output Virtualization
TOSCA	Topology and Orchestration Specification for Cloud Applications
UTS	UNIX Timesharing System
VF	Virtual Function
VM	Virtual Machine
VMM	Virtual machine manager
VNF	Virtual Network Function
VNFD	Virtual NetworkFunctions Definition
VNFM	Virtual Network Functions Manager
YAML	Ain't Markup Language

ÚVOD

Tato diplomová práce je zaměřena na jeden z moderních trendů současných počítačových sítí, a to na virtualizaci síťových prostředků, která je v současné době často diskutovaným tématem. S rozmachem cloudových řešení, a tudíž i virtualizace serverů a služeb, přestaly tradiční síťové principy založené na hardwarových prvcích vyhovovat. Začaly se tedy vyvíjet nové postupy a technologie, jak lze docílit virtualizace síťových prvků a nově také síťových služeb.

První část této práce se věnuje seznámení s možnostmi virtualizace, směry, kterými se v současnosti ubírá, a představení moderních technologií, které se využívají při virtualizaci sítě. Dále jsou představeny možnosti síťové virtualizace pro testovací, vývojové či výukové účely. Zde jsou postupně zkoumány některé nástroje od těch nejjednodušších až po propracovaná řešení pro testování produkčních systémů před nasazením. U jednodušších a středně pokročilých nástrojů jsou předvedeny konkrétní praktické ukázky.

V další části je představen framework NFV, který je standardem pro produkční nasazení virtuálních síťových prvků a služeb. Postupně jsou popsány všechny jeho součásti včetně možných konkrétních implementací systémů, které mohou být použity. Výhody a nevýhody těchto systémů jsou mezi sebou porovnány. Navíc jsou zde zdůrazněny výhody virtualizace síťových zařízení a důvody, proč je výhodné ji implementovat.

V praktické ukázce je popsán příklad detailního postupu tvorby předpisu pro automatizaci nasazení virtuální síťové služby. Tento příklad zároveň poskytuje doporučení pro tvorbu dalších předpisů pro automatizaci, a to nejen na technologiích zvolených pro tento konkrétní příklad.

Přínos této práce má být seznámení s moderními technologiemi, které jsou používány v současném rozkvětu virtuálních počítačových sítí, motivace k přechodu od tradičních fyzických řešení na virtualizovaná, představení reálných technologií, které je možné využít a předvedení nástrojů, pomocí nichž se lze s virtualizací sítě seznámit.

1 TECHNOLOGIE

Tato úvodní kapitola je věnována technologiím, které jsou stěžejní pro pochopení hlavního tématu této práce, tj. virtualizace síťových zařízení.

1.1 Virtualizace

Ačkoliv se počátky počítačové virtualizace datují do 60. let 20. století, největší rozkvět tato technologie zažívá až v posledním desetiletí. Jedná se o techniku, která emuluje fyzické výpočetní zdroje, například osobní počítače, servery, procesory, paměť, úložná zařízení, síťové prvky, samostatné aplikace nebo služby jako virtuální stroje běžící v rámci jednoho nebo více fyzických strojů, tzv. hypervizorů. Tyto virtuální stroje jsou navzájem nezávislé a oddělelené, přičemž navenek se jeví jako fyzické.[1]

Virtualizaci je možné rozdělit do následujících kategorií:

- hardwarová virtualizace,
- softwarová virtualizace,
- virtualizace na úrovni operačního systému.

1.2 Hardwarová virtualizace

Tato metoda se používala v samotných počátcích. V tomto typu virtualizace se o přidělování zdrojů virtuálním strojům staral samotný hardware, který zaručoval, že systémová volání jednotlivých virtuálních strojů se vzájemně neovlivňovala.

Z důvodu absence softwarové vrstvy se jedná o nejvýkonnější metodu. Přesto se od jejího používání velmi brzy upustilo, jelikož kladla vysoké nároky na implementaci a údržbu. V dnešní době bývá název této metody často chybně zaměňován s hardwarově asistovanou softwarovou virtualizací.[2]

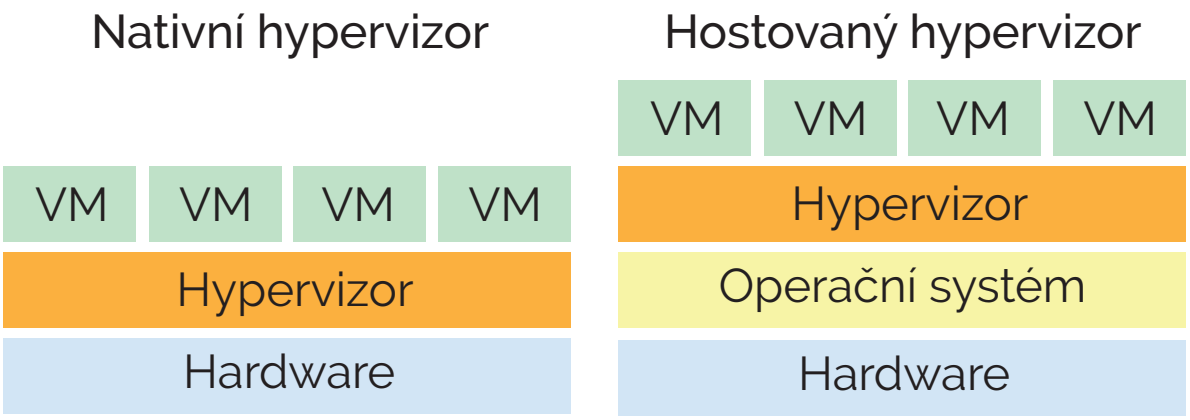
1.3 Softwarová virtualizace

Nejrozšířenější metodou je v současné době virtualizace softwarová, která se vyznačuje tím, že virtuální stroje jsou spravovány softwarovou virtualizační vrstvou. K zajištění chodu hostovaných operačních systémů je zde pro každý z těchto systémů vytvářen

virtuální hardware. V současné době se obecným pojmem virtualizace rozumí právě tato varianta.[1]

1.3.1 Hypervizor

Nejdůležitější částí softwarové virtualizace je tzv. hypervizor, někdy také nazývaný virtual machine manager (VMM). Jedná se o software, který se stará o současný běh jednoho či více virtuálních strojů a o emulaci jejich hardwarových součástí. Hypervizory lze rozlišit do dvou typů: nativní a hostovaný. Nativní hypervizor běží přímo na hostitelském stroji, obdobně jako operační systém. Hostovaný hypervizor běží jako program v rámci operačního systému hostitelského stroje.[3] Oba typy hypervizorů jsou znázorněny na obrázku č. 1.1.



Obrázek 1.1: Typy hypervizorů.

Zdroj: Vlastní (Zpracováno dle: Virtualization For Dummies [3])

1.3.2 Úplná emulace

Softwarová virtualizace má několik implementací: úplná emulace, paravirtualizace a hardwarově asistovaná virtualizace. Úplná emulace simuluje všechny hardwarové komponenty virtuálního stroje. Toho je dosaženo metodou binárního překladu systémových volání virtuálního stroje na systémová volání stroje hostitelského.[1]

Touto metodou lze zajistit běh operačního systému určeného pro konkrétní druh procesoru na hostitelském stroji s úplně odlišným procesorem. Příkladem může být OS pro procesory platformy ARM, který lze spustit na procesoru x86.[1]

Výhodou úplné emulace je vysoká flexibilita, díky které si drží místo jako volitelná možnost i na moderních virtualizačních platformách, jakými jsou například VMWare

Workstation nebo Microsoft HyperV. Nevýhodou jsou ovšem vysoké nároky na výkon, jelikož nutnost překladu všech systémových volání značně zpomaluje běh virtuálních strojů. [2]

1.3.3 Paravirtualizace

Během paravirtualizace hostovaný operační systém ví, že je virtualizovaný, a proto systémová volání provádí jiným způsobem, aby bylo možné je v hostitelském systému efektivněji zpracovat. Oproti emulaci přináší tento přístup výrazné zvýšení výkonu, pro jeho využití je ale nutné virtualizovaný operační systém upravit. V případě systému s otevřeným zdrojovým kódem, jako je Linux nebo Free BSD, se nejedná o větší problém. Ale například u operačního systému Microsoft Windows je implementace závislá na vydavateli nebo speciálních ovladačích.[4]

Paravirtualizaci podporuje v současnosti již pouze hypervizor Zen od společnosti Citrix. Její podpora byla již ukončena v linuxovém jádře v. 2.6.37 a v produktech VMWare vydaných po roce 2011.[5] [6]

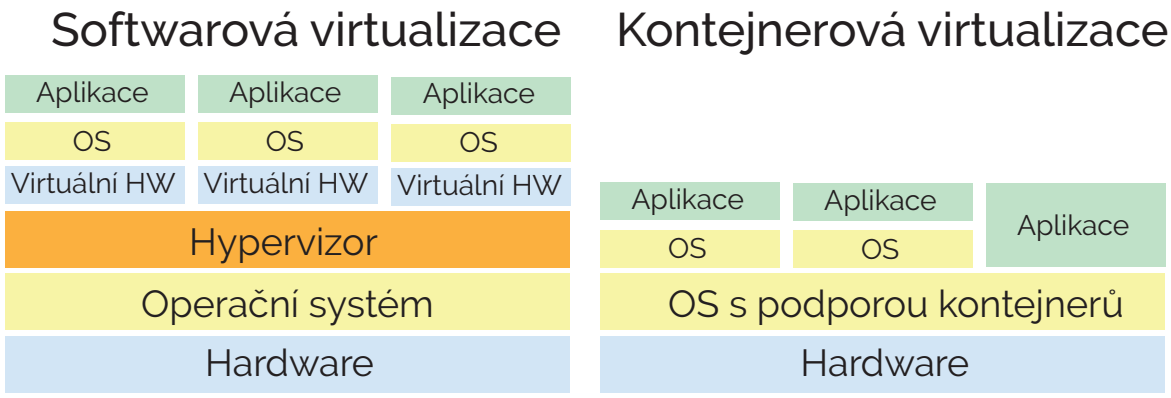
1.3.4 Hardwarově asistovaná virtualizace

Mezi lety 2006 a 2007 výrobci procesorů Intel a AMD zavedli podporu virtualizace do svých procesorů, čímž vznikla hardwarově asistovaná virtualizace. Při jejím použití může procesor zpracovávat systémová volání hostovaného operačního systému přímo, tj. bez nutnosti zásahu hostitelského operačního systému či hypervizoru. Z toho vyplývá, že zde není potřeba virtualizovat procesor, který je sdílený mezi hostitelským a hostovanými operačními systémy. Tato metoda také přináší výrazné zvýšení výkonu oproti úplné emulaci.[2]

Oproti paravirtualizaci spočívá výhoda tohoto přístupu v tom, že není nutné upravovat hostovaný operační systém. Jedinou podmínkou je podpora procesorové platformy, na které běží hypervizor. V praxi však tato podmínka nepředstavuje velké omezení, proto je tato metoda v současné době nejrozšířenější.[7]

1.4 Virtualizace na úrovni operačního systému

Tato forma je také známá pod názvem kontejnerová virtualizace. V některých publikacích se dokonce nenazývá virtualizací v pravém slova smyslu. Nenachází se zde žádná emulace fyzického hardwaru ani samostatně běžící hostované operační systémy.[8] Rozdíl mezi softwarovou a kontejnerovou virtualizací je znázorněn na obrázku č. 1.2.



Obrázek 1.2: Rozdíl mezi softwarovou a kontejnerovou virtualizací.

Zdroj: Vlastní (Zpracováno dle: blog.risingstack.com [9])

Kontejnerová virtualizace vytváří tzv. kontejnery, což jsou izolovaná virtuální prostředí v rámci hostitelského operačního systému, která s ním sdílejí systémové prostředky a jádro operačního systému. Každý kontejner má vlastní adresní prostor, souborový systém, procesy a zařízení. Obdobně jako u softwarové virtualizace se kontejner pro okolní svět tváří jako standardní fyzický stroj.[8] [9]

Pro použití kontejnerové virtualizace musí být její podpora obsažena v jádře hostitelského operačního systému a musí být nainstalovány nástroje pro správu kontejnerů. Také vytváření kontejneru se provádí odlišným způsobem než instalace operačního systému. Pro spuštění kontejneru je zapotřebí speciální šablony, která se liší v závislosti na použitém nástroji pro správu kontejnerů. Tyto šablony je možno vytvořit ručně, nebo lze použít některé již předpřipravené.[8]

1.4.1 Výhody a nevýhody kontejnerové virtualizace

Hlavní výhodou kontejnerů oproti virtuálním strojům je téměř nulová režie při jejich běhu a práci s nimi, z čehož plyne jejich vysoká rychlost a výkon. Díky tomu mají značný potenciál pro budoucí uplatnění a rozšíření do více odvětví. Technologie kontejnerové

virtualizace je ovšem stále poměrně nová a ustavičně se intenzivně rozvíjí, což s sebou přináší i různé nevýhody.[8]

Prvním nedostatkem této metody je závislost kontejnerů na jádře hostitelského operačního systému. Tato vlastnost kupříkladu neumožňuje běh linuxového kontejneru v rámci operačního systému Windows a naopak. Kombinaci více typů kontejnerů je ovšem možné provést například mezi systémy unixového typu.[9]

Hlavní a velmi závažnou nevýhodou kontejnerové virtualizace mohou být problémy s bezpečností. Z důvodu sdíleného jádra zde nedochází k úplné izolaci kontejnerů od hostitelského systému. V některých případech může tudíž dojít i k proniknutí procesu s administrátorskými právy z kontejneru do prostoru hostitelského operačního systému.[10]

1.4.2 Rozdělení kontejnerové virtualizace

V současné době lze rozdělit kontejnery na dvě skupiny: kontejnery operačních systémů, ve kterých je spuštěn celý operační systém s jednou nebo více aplikacemi, a aplikační kontejnery, které obsahují pouze jednu spuštěnou aplikaci.[9] Na obrázku č. 1.2 v pravé části lze vidět rozdíly mezi oběma typy kontejnerů.

Jako jedna z prvních kontejnerových platforem byl v operačním systému FreeBSD v roce 2000 uveden systém tzv. jails.[11] Dále následovaly kontejnery operačního systému Solaris tzv. Solaris Zones v roce 2005 a Linux Containers v roce 2008.[12] [13] V těchto platformách jsou využívány kontejnery operačních systémů. Nyní se ve značné míře rozmáhají aplikační kontejnerové platformy, jejichž nejvýznamnějšími zástupci jsou Docker a Rocket.

Microsoft ve svých systémech Windows 10 a Server 2016 také nově zavedl podporu vlastních aplikačních kontejnerů, které mohou být spravovány kromě Powershellu i nástrojem Docker. Microsoft zároveň představil Hyper-V Container, což je odlehčený virtuální stroje hostující pouze kontejner. Tato kombinace má řešit problémy s bezpečností kontejnerů, jako je například již zmíněná eskalace procesu s vyššími oprávněními do hostujícího operačního systému.[14]

1.5 Softwarově definované sítě

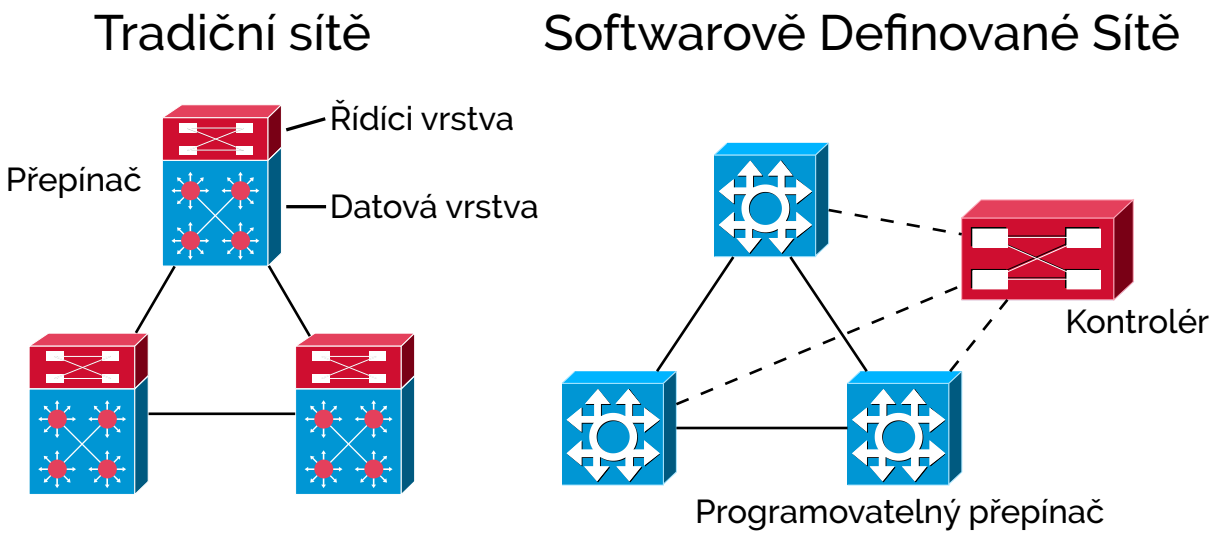
Logika při přeposílání paketů na jednotlivých síťových zařízeních se skládá ze dvou částí: řídicí logiky a datové logiky. Řídicí logika rozhoduje, kterým směrem budou pakety přeposílány a jakou budou mít prioritu, tato funkcionalita je velmi komplexní a klade vyšší nároky na výkon příslušného hardwaru.[15]

Datová logika již není tak komplexní, vykonává pouze přeposílání paketů na základě pravidel poskytnutých řídicí logikou, proto nevyžaduje příliš výkonný hardware. Tato pravidla se ve světě softwarově definovaných sítí nazývají flow pravidly.[15]

1.5.1 Srovnání tradiční sítě a SDN

V tradičním přístupu k počítačovým sítím jsou obě tyto funkce obsaženy v každém síťovém prvku. Tento přístup vyžaduje, aby všechny spolupracující prvky využívaly stejné řídicí protokoly, což je z pohledu nasazení, správy a rozšiřitelnosti velmi neflexibilní.[15]

Softwarově definované síťě ovšem tento přístup mění. Řídicí funkce síťových zařízení je soustředěna centrálně v jednom nebo více kontrolérech a na ostatních síťových prvcích zůstává pouze funkce datová. Toto řešení přináší zjednodušení nasazení a správy sítě a zvyšuje flexibilitu při rozšiřování nebo restrukturalizaci sítě.[15] Rozdíl mezi tradiční sítí a SDN je znázorněn na obrázku č. 1.3.



Obrázek 1.3: Rozdíl mezi tradiční a softwarově definovanou sítí.

Zdroj: Vlastní (Zpracováno dle: Foundations of Modern Networking [15])

1.5.2 Princip fungování

Z důvodu složitosti a výkonové náročnosti řídicí logiky je vhodné ji přesunout z velkého množství síťových prvků do centrálního kontroléru a snížit tak nároky na jejich výkon. Kontrolér sice stále požaduje velmi výkonný hardware, ten je ovšem sdílen více síťovými prvky, a tudíž se celkové nároky na výkon snižují.[15] [16]

Každý datový tok v softwarově definované síti je řízen kontrolérem, který určí, zda může být povolen na základě bezpečnostních politik, a následně spočítá cestu, kterou budou jeho pakety přeposílány. Poté umístí flow pravidla na síťové prvky, přes které tento tok povede, ty na základě těchto pravidel provedou samotné přeposílání paketů a uskutečnění komunikace, případně blokování tohoto provozu.[15]

Flow pravidla jsou ukládána v paměti síťového prvku v tzv. flow tabulce. Zde se při výchozím nastavení vyskytují krátkou dobu, kterou určuje kontrolér na základě doby uplynulé od posledního použití těchto pravidel. Při příchozím paketu na síťový prvek je nejprve hledáno odpovídající pravidlo ve flow tabulce. Pokud žádné z nich není schopné určit, jak paket zpracovat, síťový prvek požádá kontrolér o poskytnutí nového pravidla pro tento paket. Pro komunikaci mezi síťovými prvky a kontrolérem se v dnešní době nejčastěji používá protokol OpenFlow.[16]

1.6 Jmenné prostory v Linuxu

Jmenné prostory vytváří abstrakci globálních systémových zdrojů, díky které se tyto virtuální zdroje uvnitř jmenného prostoru jeví jako globální. Jakékoliv změny systémových zdrojů uvnitř jmenného prostoru nejsou znatelné mimo něj. Jak je jistě patrné z předchozí kapitoly, jmenné prostory jsou využívány jako implementace linuxových kontejnerů. V Linuxu existuje šest typů jmenných prostorů:

- Uživatelské prostory,
- prostory procesů (PID),
- prostory meziprocesové komunikace (IPC),
- síťové prostory,

- prostory kořenových adresářů,
- prostory doménových názvů (UTS).

[17]

Uživatelské jmenné prostory izolují bezpečnostní identifikátory a atributy, jako jsou například identifikátory uživatelů a skupin, klíče a oprávnění. Tyto identifikátory a parametry mohou být pro jeden proces odlišné v různých uživateských jmenných prostorech. To v praxi zpravidla znamená, že proces může běžet v jednom uživatelském prostoru s právy standardního uživatele a v jiném prostoru s administrátorskými právy.[17]

Prostory procesů oddělují identifikátory procesů (PID). To znamená, že různé procesy mohou mít v oddělených prostorech procesů stejný identifikátor. Identifikátory procesů uvnitř každého prostoru procesů začínají od čísla 1 a nové procesy vznikají uvnitř stejného jmenného proscesu systémovým voláním fork, stejně jako je tomu při startu operačního systému unixového typu. Identifikátory procesů jsou uvnitř jmenného prostoru procesů unikátní.[17]

Jmenné prostory meziprocesové komunikace (IPC) oddělují zdroje meziprocesové komunikace, především objekty IPC Systému V a posixové fronty zpráv. V každém prostoru IPC používají tyto objekty jiný způsob identifikace než cestu v souborovém systému.[17]

Síťové jmenné prostory izolují zdroje týkající se síťové komunikace, jako jsou síťová rozhraní, protokoly IPv4 a IPv6, směrovací tabulky, firewally, síťové adresáře, sockety atd. Každé fyzické síťové rozhraní může patřit pouze do jednoho síťového jmenného prostoru. Pro komunikaci mezi síťovými jmennými prostory se dají použít páry virtuálních síťových rozhraní, které vytváří virtuální síťové tunely mezi těmito jmennými prostory. Ve výsledku se tyto tunely chovají jako fyzické kabelové linky. Po odstranění síťového jmenného prostoru jsou jeho fyzická rozhraní přesunuta do výchozího síťového jmenného prostoru.[17]

Procesy uvnitř různých jmenných prostorů kořenových adresářů vidí odlišnou adresářovou hierarchii souborového systému. Jmenné prostory doménových názvů poskytují izolaci dvou systémových identifikátorů: názvu hosta a doménového jména NIS.

Z toho vyplývá, že doménový název počítače je v různých jmenných prostorech UTS odlišný.[17]

1.7 Cloud

Cloud computing je v IT stále poměrně nový pojem, začal se používat teprve kolem roku 2007. Jedná se o model, který umožňuje všudypřítomný, spolehlivý a jednoduchý přístup ke sdíleným, plně konfigurovatelným výpočetním zdrojům, jako jsou například: servery, úložná zařízení, síť, aplikace nebo služby.[18]

1.7.1 Základní charakteristiky

- Výpočetní prostředky na požádání:
  Uživatel je schopen si sám zajistit požadované výpočetní zdroje bez nutnosti interakce s poskytovatelem služeb.[18]
- Přístup přes síť:
  Všechny prostředky jsou dostupné přes počítačovou síť a jednoduše spravovatelné prostřednictvím standardních přístupů například přes webovou nebo desktopovou aplikaci či programové rozhraní.[18]
- Shromažďování prostředků:
  Posktovatel cloudu má typicky některé zdroje připravené předem. Tyto jsou dynamicky přidělovány a odebírány jendotlivým uživatelům podle aktuální potřeby.[18]
- Dynamické škálování:
  Jedná se o automatické rozšiřování nebo snižování aktuálně přidělených zdrojů na základě jejich okamžité vytíženosti.[18]
- Měřitelné služby:
  Rozsah a vytížení využívaných zdrojů jsou monitorovány a tyto výsledky jsou poskytnuty uživateli i poskytovateli například pro účtování nebo informační účely.[18]

1.7.2 Modely služeb

- Software as a Service (SaaS):
  Schopnost poskytovat uživateli aplikace vytvořené poskytovatelem běžící na jeho

cloudové infrastruktuře. Aplikace mohou být přístupné z různých zařízení buď přes webový prohlížeč nebo aplikační programové rozhraní.[19]

- Platform as a Service (PaaS):
  Schopnost poskytovat uživateli prostředí pro své aplikace. Zákazník nespravuje nižší vrstvy cloudové infrastruktury, jako je síť, servery, nebo úložiště, ale pouze nastavení pro hostování svých aplikací.[19]
- Infrastructure as a Service (IaaS):
  Schopnost poskytovat uživateli výpočetní výkon, úložiště, síťové a další zdroje, na kterých je uživatel schopen provozovat svůj software.[19]

1.7.3 Rozdělení cloudových platforem

- Veřejný cloud:
  Cloudová infrastruktura je nabízena veřejnosti, typicky za poplatek.[19]
- Privátní cloud:
  Cloudová infrastruktura je využívána pouze společností, která ji provozuje.[19]
- Comunity cloud:
  Cloudová infrastruktura je nabízena pouze pro specifickou komunitu uživatelů z organizací, které mají sdílený koncern.[19]
- Hybridní cloud:
  Jedná se o kombinaci dvou a více výše uvedených druhů cloudu.[19]

2 TESTOVACÍ A VÝUKOVÁ PROSTŘEDÍ PRO VIRTUALIZACI SÍTĚ

Tato kapitola se bude věnovat možnostem virtualizace síťových zařízení pro vývoj, testování nebo výukové účely. Postupně budou představeny nástroje od těch nejjednodušších informačních a výukových až po profesionální, které se využívají pro analýzu a přípravu produkčních prostředí.[20]

2.1 Mininet

Mininet je emulátor počítačových sítí, který je schopen vytvářet virtuální koncové stanice, virtuální přepínače, virtuální linky a kontroléry softwarově definovaných sítí. Je dostupný pouze pro operační systém Linux, na jehož jádře je závislý. Virtuální přepínače, které vytváří, podporují protokol OpenFlow, pomocí něhož kontrolery softwarově definovaných sítí ovládají přepínací pravidla v přepínačích.[20]

2.1.1 Princip fungování

Jedná se o velmi jednoduchý a výkonný nástroj určený především pro výzkum, vývoj, testování či výuku softwarově definovaných sítí, jelikož se soustředí především na virtualizaci přepínačů Open vSwitch a na protokol OpenFlow. Ve své oblasti ovšem poskytuje vysoký výkon s velmi malou náročností na systémové zdroje. Bylo otestováno, že dokáže spustit až 4096 instancí virtuálního přepínače na jednom fyzickém počítači. I přesto není určen pro běh v produkčních prostředích.[20]

Mininet je vyvíjen komunitou jako software s otevřeným zdrojovým kódem dostupným na Githubu. [21] Jeho převážná část je napsaná v programovacím jazyce Python krom některých utilit napsaných v jazyce C. Navíc poskytuje aplikační programové rozhraní (API) pro Python, díky kterému je snadno rozšiřitelný o nové specifické moduly. [20]

Pro vytváření virtuálních přepínačů využívá jmenné prostory operačního systému Linux nebo linuxové kontejnery LXC, pokud jsou dostupné. Minimálním systémovým požadakem je pouze linuxové jádro verze 2.2.26, které podporuje jmenné prostory. Pro každou koncovou stanici je vytvořen zvláštní uživatelský jmenný prostor nebo kontejner LXC. Přepínače jsou typycky umístěny v hlavním jmenném prostoru. Pouze při

použití přepínačů User-space switch je možné tyto umístit do oddělených jmenných prostorů. Při použití přepínačů Open vSwich toto není umožněno. Všechny virtuání přepínače a koncové stanice jsou propojeny virtuálními ethernetovými linkami (veth).[20]

2.1.2 Praktické využití

Největší přednosti Mininetu jsou jeho snadná instalace a velice jednoduché ovládání. Možnosti instalace jsou tři: předpřipravený virtuální stroj s operačním systémem Ubuntu a nainstalovaným Mininetem, který lze spustit na téměř jakémkoli současném hypervizoru; instalace jako balíčku na operačním systému Ubuntu, v jehož repozitářích je standardně dostupný, nebo stažení zdrojového kódu z Githubu a jeho následná kompilace.[22]

Po instalaci je mininet okamžitě připraven ke běhu bez nutnosti dodatečné konfigurace. Pro prvotní otestování stačí spustit následující příkaz, který automaticky vytvoří jednoduchou síťovou topologii skládající se z jednoho přepínače a dvou koncových stanic.[20]

```
$ sudo mn
```

Existují dva způsoby, jak lze vytvářet síťové topologie v Mininetu. První způsob je určení typu a velikosti topologie pomocí jejího názvu a čísla předaných v parametru při spouštění Mininetu. Základní topologie při spuštění bez parametru byla již popsána. Dále lze vybrat ze třech topologií: single, tree a linear.[20] Na obrázku č. 2.1 jsou tyto topologie znázorněny.

Topologie single vytvoří jeden přepínač a počet hostů N, přičemž tento počet je předán v parametru. Všechny koncové stanice jsou připojeny do tohoto přepínače.[20]

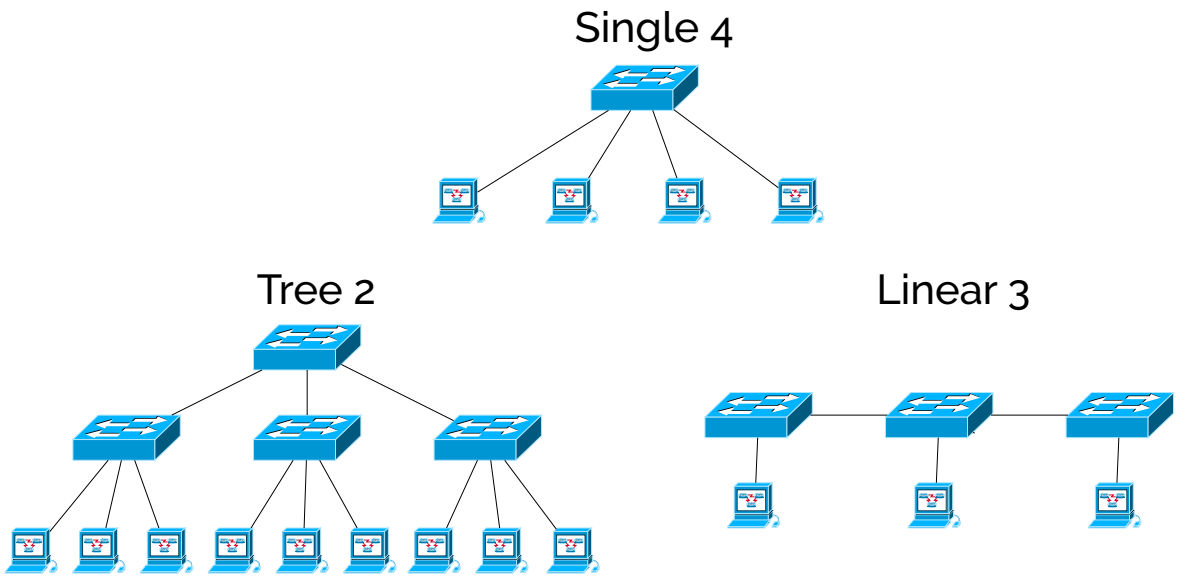
```
$ sudo mn --topo single,4
```

Topologie linear vytvoří N přepínačů a N hostů. Přepínače jsou propojeny v řadě za sebou a ke každému z nich je připojen jeden host.[20]

```
$ sudo mn --topo linear,4
```

Na závěr topologie tree vytvoří z přepínačů strom o výšce N, v jehož listech se vyskytují koncové stanice.[20]

```
$ sudo mn --topo tree,4
```



Obrázek 2.1: Výchozí topologie v Mininetu.

Zdroj: Vlastní (Zpracováno dle: slideshare.net [23])

Kromě těchto základních topologií lze samozřejmě definovat i vlastní a daleko specifičtější, které se vytvářejí jako třídy v jazyce Python. Zde lze kromě uspořádání definovat také různé charakteristiky linek, jako je např. propustnost nebo zpoždění. Následující kód obsahuje krátkou ukázku vlastní topologie.[24]

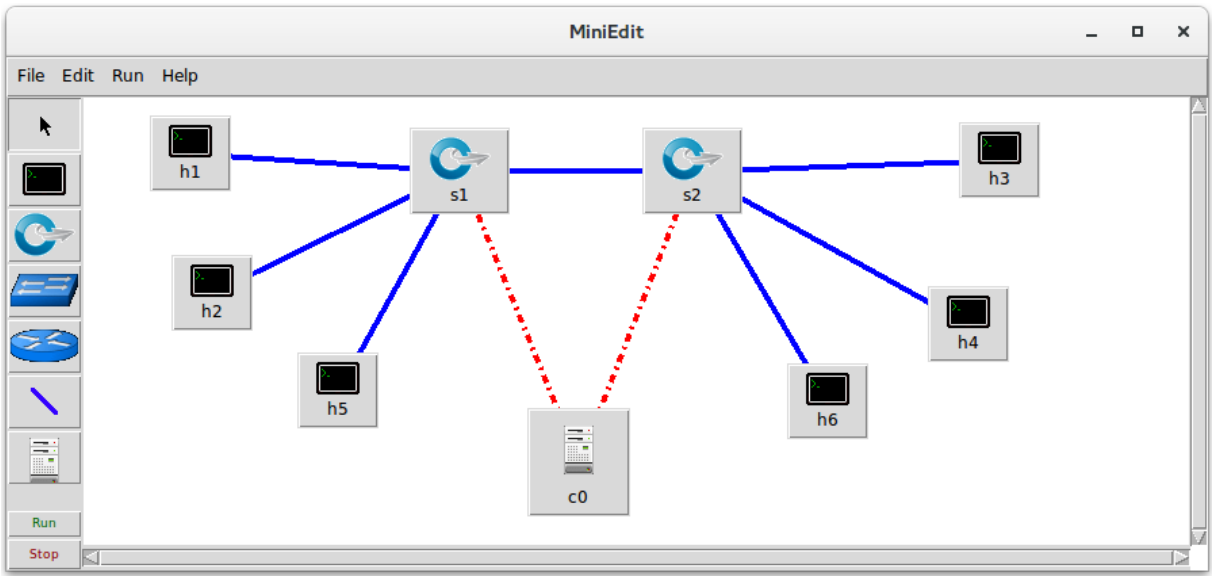
```
from mininet.topo import Topo
class MyTopo( Topo ):
  def __init__( self ):
      Topo.__init__( self )
      leftHost = self.addHost( 'h1' )
      rightHost = self.addHost( 'h2' )
      leftSwitch = self.addSwitch( 's3' )
      rightSwitch = self.addSwitch( 's4' )
      self.addLink( leftHost, leftSwitch )
      self.addLink( leftSwitch, rightSwitch )
      self.addLink( rightSwitch, rightHost )
```

```
topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Na Mininet lze napojit jakýkoliv kontrolér SDN, který podporuje protokol Open Flow. Určení, který kontrolér se má použít, se provede nastavením příslušného parametru na jeho adresu IP a port při spouštění Mininetu. Pokud se žádný kontrolér nespecifikuje, je použit výchozí kontrolér NOX, který je obsažen v Mininetu.[24] Následující příkaz znázorňuje spuštění Mininetu se specifickým kontrolérem.

```
$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
```

Pro uživatele, kteří nemají zkušenosti s programováním v jazyce Python, nabízí Mininet grafickou utilitu s názvem Miniedit, ve které je možné sestavit vlastní topologii umísťováním síťových prvků na pracovní plochu a následným propojením virtuálními linkami pouhým přetažením myši. Tento nástroj při exportu topologie vygeneruje příslušný kód v jazyce Python, odpovídající graficky navržené topologii.[25] Náhled pracovního prostředí nástroje Miniedit je znázorněn na obrázku č. 2.2



Obrázek 2.2: Pracovní prostředí MiniEdit.

Zdroj: Vlastní

2.1.3 Shrnutí

Mininet se skvěle hodí například začátečníkům v oblasti SDN nebo vývojářům kontrolérů pro SDN z důvodů jeho jednoduchosti a nízké náročnosti na výkon. Jelikož dokáže virtualizovat přepínače na úrovni kontejnerů, které se chovají vůči kontroléru

téměř stejně jako fyzické prvky, je možné snadno přenést kontrolér navržený a otestovaný na Mininetu i na jiné virtualizované, nebo dokonce fyzické prostředí.

2.2 Virtuální síť na hypervizoru

Tato kapitola je zaměřena na vytváření virtuálních síťových zařízení a linek tzv. "na vlastní pěst"za pomoci hypervizoru pro virtualizaci koncových a síťových zařízení a nástrojů operačního systému pro vytváření virtuálních linek.

2.2.1 Operační systémy sítových prvků

Síťová zařízení jsou v podstatě také počítače, na kterých běží operační systém. Jejich hardware se stejně jako u běžných stolních počítačů nebo serverů skládá z procesoru, operační paměti, úložiště a především z rychlých síťových rozhraní. Největší rozdíl spočívá v použitém operačním systému, který je na síťových zařízeních optimalizován pouze pro úkony týkající se síťové komunikace.[26]

Na rozdíl od desktopových nebo serverových operačních systémů, na kterých jsou provozovány uživatelské aplikace různých druhů, se operační systémy na síťových prvcích soustředí pouze na služby síťové komunikace, jako je například: přepínání paketů, směrování, firewall, rozdělování zátěže atd. Často bývá operační systém těchto zařízení určen pouze pro malou část, nebo dokonce jen jednu z těchto rolí.[26]

Jelikož se virtualizace serverových a desktopových operačních systémů za poslední léta posunula velice dopředu, přinesla značná zlepšení především v oblasti výkonu virtuálních strojů a snížil se také celkový úbytek výkonu při jejím použití, začaly se rozvíjet také operační systémy pro virtuální síťová zařízení, kde je dostatečný výkon kritický pro rychlý chod sítě.

Vlastní operační systémy pro virtuální síťová zařízení nabízejí jak velcí hráči na poli síťových technologií Cisco a Juniper, tak různé menší společnosti a komunity pro vývoj otevřeného softwaru. Cisco nabízí operační systém pro virtuální směrovač CSR 1000v a virtuální firewall ASAv.[27] [28]

Juniper obdobně nabízí virtuální formu svého přepínače MX zvanou vMX a taktéž vSRX jakožto virtuální firewall.[29] [30] Z otevřených řešení se nabízí například VyOS jako opensourcová varianta proprietárního vRouteru od společnosti Brocade (dříve vlast-

něného společností Vyatta, odtud název VyOS) nebo systém OpenWRT pro směrovač do domácností nebo malých firem.[31] [32] [33]

2.2.2 Instalace

Při tomto přístupu experimentování s virtuální sítí má její návrhář prakticky volnou ruku, je omezen pouze hardwarovými zdroji, případně licencemi pro použití proprietárního softwaru. Důležité je také zvolit správné technologie, které jsou kompatibilní s použitou platformou. Některé nástroje jsou kompatibilní napříč více platformami, jako například VMWare Workstation nebo Open vSwitch. Jiné jsou určené pouze pro specifickou platformu, jakou je například KVM-QEMU v Linuxu nebo proprietárni hypervizor Hyper-V pro Microsoft Windows.

Jako nástroje pro následující ukázku byly zvoleny tyto technologie: hostitelský operační systém Linux v distribuci Ubuntu 16.04, KVM-QEMU pro virtualizaci koncových stanic a směrovačů, Open vSwich pro vytváření virtuálních přepínačů a technologie virtuálních ethernetových linek obsažená v Linuxu.

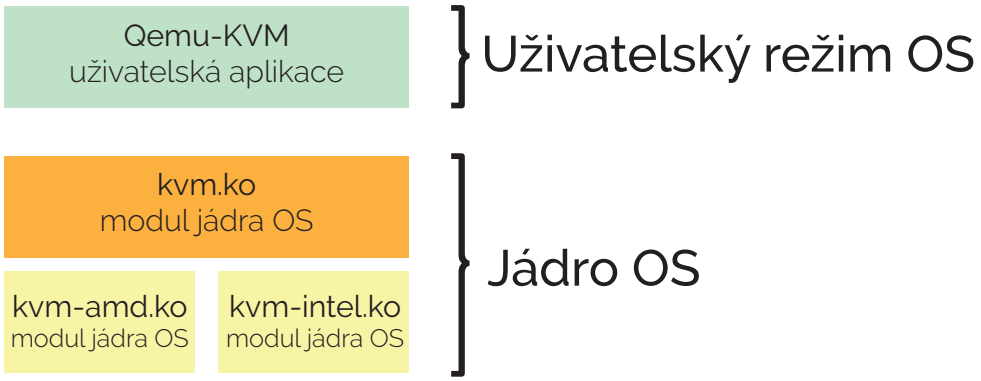
KVM-QEMU

Jak již bylo zmíněno v kapitole 1.3.1, hypervizory se dělí na dva druhy: bežící pod operačním systémem a běžící namísto operačního systému. Technologie Kernel-based Virtual Machine (KVM) ale spadá do obou těchto kategorií. Po instalaci KVM je do jádra operačního systému Linux zaveden modul kvm.ko, jenž ho přemění v hypervizor, který je schopen spouštět virtuální stroje jako samostatné linuxové procesy.[34] [2]

Jádro systému je ovšem schopno spravovat pouze některé systémové prostředky, proto je nutné použít uživatelský program QEMU, který zajišťuje hardwarovou emulaci a správu virtuálních strojů. O operačním systému Linux s podporou KVM v jádře a ve spolupráci s programem QEMU již lze hovořit jako o hypervizoru.[34]

Technologie KVM podporuje pouze jeden způsob softwarové virtualizace, kterým je hardwarově asistovaná virtualizace. Z toho důvodu je k provozování KVM-QEMU kromě operačního systému Linux nutný také procesor, který ji podporuje.[34]

Při startu KVM je kromě modulu kvm.ko do jádra zaveden také jeden z následujících dvou modulů. V případě, že procesor hostitelského počítače je od firmy Intel, je načten modul kvm-intel.ko. Pokud je však procesor značky AMD, zavádí se modul kvm-amd.ko. [34] Na obrázku č. 2.3 je zobrazen přehled implementace KVM-QEMU.



Obrázek 2.3: Architektura KVM-QEMU.
Zdroj: Vlastní (Zpracováno dle: Mastering KVM Virtualization [34])

Jelikož KVM provádí virtualizaci na úrovni jádra hostitelského operačního systému, QEMU se stará o správu virtualizace v uživatelském prostředí a navíc lze operační systém nadále využívat i k jiným účelům, než je virtualizace, nelze tuto technologii zařadit ani do jedné z kategorií hypervizorů. Tato vlastnost se skvěle hodí, pokud je potřeba vysoký výkon virtualizovaných zařízení a zároveň jsou na hostitelský systém kladeny další specifické požadavky, které běžný bare-metal hypervizor nepodporuje. Příkladem může být cloudová platforma OpenStack, která se využívá při provozu privátních cloudových datových center, ta pro virtualizaci používá právě hypervizor KVM-QEMU.[34] Příklady využití OpenStacku budou podrobněji popsány v následující kapitole.

Open vSwitch

Open vSwitch je virtuální multilayerový přepínač vyvíjený jako software s otevřeným zdrojovým kódem pod licencí Apache 2.0. Je velmi snadno programově rozšiřitelný a podporuje širokou škálu automatizačních metod. Kromě standardních funkcí přepínače podporuje také velké množství dalších síťových protokolů, např. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag.[35]

Kromě virtuální implementace je také možné Open vSwitch použít jako řídicí vrstvu v hardwarových přepínačích. Je neustále aktivně vyvíjen a rozšiřován o podporu nových technologií, jako je v současné době například sada programových knihoven a ovladačů pro rychlé zpracování paketů nazývaná DPDK. Díky svému vysokému výkonu je velice často provozován v produkčních prostředích pro přepínání paketů mezi virtuálními stroji.[36][37]

Příklad

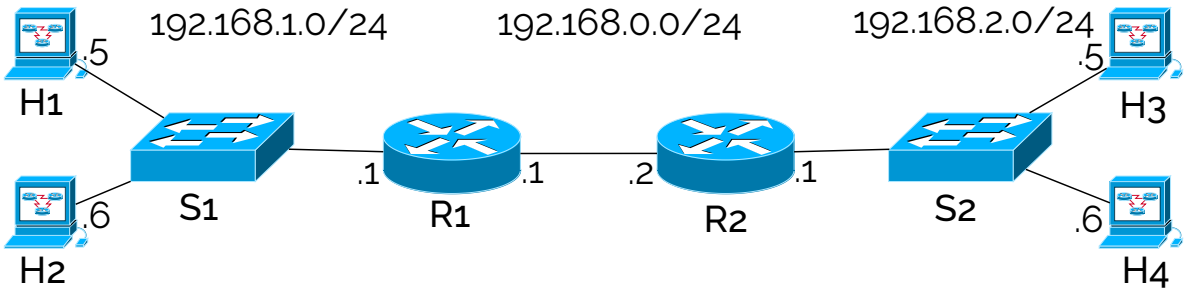
Následující ukázková síťová topologie se skládá ze dvou virtuálních směrovačů, dvou virtuálních přepínačů a čtyřech koncových stanic připojených v odlišných podsítích. Schéma zapojení zachycuje obrázek č. 2.4. Pro běh směřovače R1 byl zvolen operační systém VyOS ve verzi 1.1.7, který je postavený na základě linuxové distribuce Ubuntu jako otevřený operační systém pro virtuální směrovače. Jeho konfigurační rozhraní se velmi podobá rozhraní operačního systému Junos pro síťová zařízení od firmy Juniper. Je dostupný zdarma ke stažení na oficiálních stránkách vyos.io.[38]

Jako operační systém pro směrovač R2 byl vybrán operační systém IOS běžící ve virtuálním směrovači CSR 1000v. Směrovač 1000v je proprietární řešení od společnosti Cisco, komerčně dostupné pouze pod placenou licencí s plnou podporou. Nicméně firma Cisco poskytuje některé starší verze k dispozici zdarma pro testování a výuku. Lze je po regisraci stáhnout na oficiálních stránkách cisco.com, kde je také dostupný oficiální návod na instalaci.[27]

Na koncových stanicích není nutno používat žádný náročný a velký operační systém, proto byl pro ně zvolen nenáročný systém CirrOS, jenž je vyvíjen pro účely testování. Jedná se o velmi jednoduchou linuxovou distribuci, která se vyznačuje malou velikostí (cca 12 MB) a rychlým spouštěním, protože obsahuje jen ty nejzákladnější programy a služby nutné pro otestování síťové funkcionality.

Instalace

Následující instalační postup předpokládá, že je již nainstalovaný hostitelský operační systém Ubuntu, který je již připravený k používání. Instalace potřebných součástí pro následující příklad je velmi jednoduchá, stačí spustit následující příkaz, který nainstaluje hypervizor KVM-QEMU a Open vSwitch.



Obrázek 2.4: Topologie ukázkové virtualizované sítě.

Zdroj: Vlastní

```
$ sudo apt install qemu-kvm libvirt-bin bridge-utils virt-manager
   openvswitch-switch
```

Před spuštěním virtuálních strojů koncových stanic a směrovačů je vhodné si předem připravit přepínače, jejich porty a páry virtuálních ethernetových portů. Po provedení následujícího příkazu se vytvoří prázdný přepínač Open vSwich s názvem S1. Obdobným způsobem byl vytvořen i přepínač S2.

```
$ sudo ovs-vsctl add-br S1
```

Takto vytvořený přepínač neobsahuje žádné porty. Zavoláním příkazu níže se do přepínače S1 přidá port s názvem S1H1. Pokud je v systému nalezeno síťové rozhraní s tímto názvem, ať už fyzické či virtuální, tak zde není nutno uvádět typ internal, rozhraní se pouze přiřadí uvedenému přepínači jako standardní typ. Open vSwitch standardně očekává, že do něj budou přiřazována již existující rozhraní, specifikováním typu internal se vytvoří virtuální rozhraní svázané s tímto přepínačem, které je při jeho odstranění odstraněno také. Tímto způsobem byla vytvořena i zbývající rozhraní.

```
$ sudo ovs-vsctl add-port S1 S1H1 -- set Interface S1H1
   type=internal
```

Nyní jsou připraveny všechny porty přepínačů pro koncová zařízení a směrovače, zbývá tedy vytvořit pár virtuálních ethernetových linek, který bude propojovat směrovače. Tento pár se chová jako fyzický síťový kabel propojující dvě zařízení. Vytvořená rozhraní dostanou názvy R1R2 a R2R1. Jeho vytvoření se provede následujícím příkazem.

```
$ ip link add dev R1R2 type veth peer name R2R1
```

Posledním krokem instalace virtuálního prostředí je spuštění virtuálních strojů. To lze provést příkazy uvedenými níže. V něm jsou specifikovány hardwarové požadavky stroje. Parametr '–import' udává, že pevný disk virtuálního stroje je již vytvořený a nainstalovaný v souboru předaném jako cesta do parametru '–disk' a není nutné ho vytvářet a instalovat operační systém. Parametr '–network' specifikuje virtuální síťovou kartu, která je zde zapojena do odpovídajícího rozhraní přepínače nebo virtuálního portu. Virtuální koncové stanice lze vytvořit pomocí tohoto příkazu.

```
$ sudo virt-install \
  --name H1_Cirros \
  --ram 1024 \
  --disk path=./cirros-3.5.1-H1.qcow2 \
  --import \
  --vcpus 1 \
  --os-type Linux \
  --os-variant generic \
  --network bridge=S1H1 \
  --graphics vnc,port=5999 \
  --console pty,target_type=serial
```

Vytvoření směrovače se liší především v použití dvou síťových rozhraní. U směrovače Cisco 1000v je důležité mu přiřadit alespoň 4 GB operační paměti, kvůli jejímu nedostatku se operační systém IOS nedokáže spustit.

```
$ sudo virt-install \
  --name R2-CSR1000v \
  --ram 4096 \
  --disk
     path=./csr1000v-universalk9.03.15.00.S.155-2.S-std-R2.qcow2 \
  --import \
  --vcpus 2 \
  --os-type generic \
  --os-variant generic \
```

```
  --network bridge=S1R1 \
  --network bridge=R1R2 \
  --graphics vnc,port=5999 \
  --console pty,target_type=serial \
```

Po spuštění všech instancí virtuálních strojů je infrastruktura tohoto prostředí připravena. Pro úplnou funkčnost je třeba nakonfigurovat všechny koncové stanice a síťové prvky. Koncovým stanicím stačí nastavit adresu IP a u směrovačů je nutné kromě konfigurace adres IP nastavit také směrování. V tomto příkladu byl pro směrování využit protokol OSPF. Kompletní konfigurace všech koncových zařízení a směrovačů jsou z důvodu jejich obsáhlosti obsaženy v přílohách A a B .

Nyní příkaz pro výpis všech virtuálních strojů, jehož výstup znázorňuje následující ukázka, která říká, že všechny definované virtuální stroje jsou běžící. Nyní je tedy možné začít testovat konektivitu a provádět konfiguraci.

```
$ virsh list --all
Id    Name                           State
----------------------------------------------------
-     H1_Cirros                      running
-     H2_Cirros                      running
-     H3_Cirros                      running
-     H4_Cirros                      running
-     R1_VyOS                        running
-     R2_CSR1000v                    running
```

Z následujícího výstupu příkazu ping, spuštěného na stanici H1, který testuje konektivitu mezi ní a sanicí H3, je patrné, že komunikace funguje bez problémů, tím pádem je funkční i směrování. Všechny výstupy příkazu ping jsou obsaženy v přílohách C – F .

```
$ ping 192.168.2.5
PING 192.168.2.5 (192.168.2.5): 56 data bytes
64 bytes from 192.168.2.5: seq=0 ttl=62 time=2.056 ms
64 bytes from 192.168.2.5: seq=1 ttl=62 time=2.888 ms
64 bytes from 192.168.2.5: seq=2 ttl=62 time=2.684 ms
64 bytes from 192.168.2.5: seq=3 ttl=62 time=3.246 ms
```

Shrnutí

Tento přístup vytváření virtuální síťové infrastruktury má výhodu úplné kontroly nad všemi jejími komponentami a velké volnosti při výběru použitých technologií. Například je možno kombinovat virtuální stroje s kontejnery, využívat různé technologie virtuálních přepínačů či datových linek, nebo lze tuto virtuální infrastrukturu spojit s fyzickými komponenty.

Nicméně pro rozsáhlé síťové infrastruktury, které se velice často vyskytují v produkčních prostředích, již takovéto řešení nepřináší mnoho výhod oproti fyzickým zařízením. Velký počet virtuálních prvků a linek je náročný na správu, dále na detekci a opravu případných chyb. Při nutnosti použití většího množství služeb kromě přepínání a směrování paketů, například připojování různých typů externích úložišť, pokročilého filtrování paketů nebo dynamického rozdělování zátěže, je zapotřebí použít velké množství různých technologií, jejichž správa je nejednotná, přičemž každou z nich je třeba znát velice podrobně.

Tohoto přístupu se dá využít například při výuce nebo vývoji a testování nových technologií a postupů. Velké cloudové virtualizační platformy, jako je například Openstack nebo OpenShift, jsou ve skutečnosti souhrnem několika menších technologií pro specifické účely pod jednotnou správou a dohledem.

2.3 Juniper Junosphere

Junosphere Cloud je komerční nástroj poskytovaný společností Juniper. Jedná se o službu, která umožňuje síťovým architektům a expertům vytvářet stabilní virtuální síťová prostředí hostovaná na serverech Juniperu.[39]

2.3.1 Představení služeb

Uživatelé si tedy pronajímají prostředí pro provoz virtuálních síťových zařízení jako službu. Nemusí tedy vlastnit žádnou fyzickou infrastrukturu ani speciální software, k prostředí je možný přístup pomocí webové aplikace, nebo aplikačního programového rozhraní.[39]

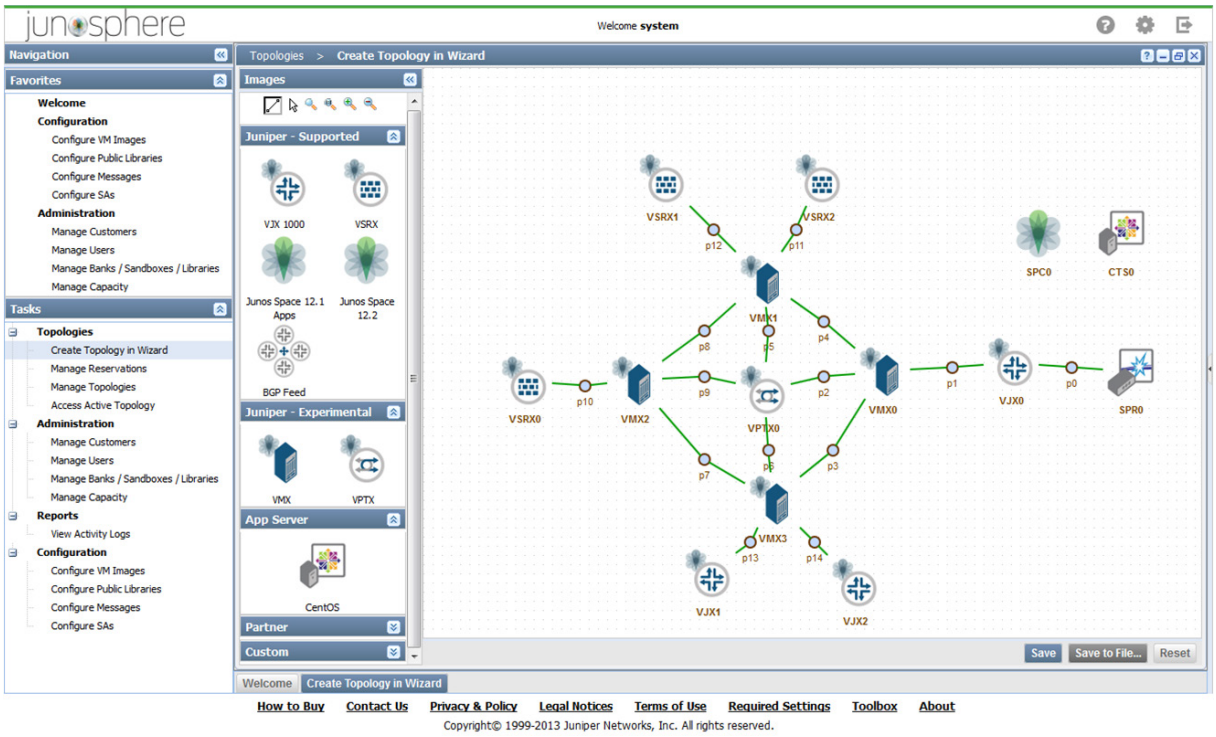
Jelikož se jedná o proprietární řešení společnosti Juniper, veškeré virtuální síťové prvky jsou založené na operačním systému Junos od této firmy. Velkou výhodou je ovšem

přesná simulace téměř všech fyzických síťových zařízení od Juniperu, a to včetně výpočetního výkonu a propustnosti. Tuto funkcionalitu zajišťuje speciální hypervizor, který je také proprietárně vyvinutý firmou Juniper speciálně pro Junosphere.[39]

2.3.2 Uživatelské rozhraní

Uživatelské rozhraní je velmi propracované a nabízí širokou škálu možností, které lze při vývoji, testování či školení využít. Kromě možnosti grafického návrhu celkové síťové topologie nabízí také plný přístup ke všem rozhraním všech síťových prvků, ať už grafickým, textovým či aplikačním.[39]

Navíc lze poskytnout přístup k virtuální síti například z externí podnikové sítě či dokonce veřejně. Umožňuje správu uživatelů a oprávnění, to lze využít například při podnikových školeních. Dále lze nastavit přístupové doby pro jednotlivé uživatele či naplánovat časový plán provozu jednotlivých zařízení.[39] Ukázka grafického rozhraní je na obrázku č. 2.5



Obrázek 2.5: Grafické rozhraní Juniper Junosphere.

Zdroj: Junosphere Cloud Datasheet [39]

2.3.3 Nadstandardní služby

Junosphere nabízí také možnost připojení virtuální sítě k fyzické podnikové síti pomocí technologie nazývané Junosphere Connector. Ta umožňuje napojení k fyzické síti, kde nejsou síťové prvky od Juniperu, tím lze otestovat kompatibilitu sítě s prvky od jiných výrobců či dokonce s otevřenými technologiemi.[39]

Juniper nabízí dva typy financování. Jedná se buď o tzv. pay-as-you-go, čili placení pouze za využitý výkon, nebo o roční předplatné s několika úrovněmi omezení prostředků. První možnost se vyplatí především zákazníkům, kteří využijí službu pouze krátkodobě s nízkými nároky na výkon a kapacitu. Předplatné je zase atraktivní pro ty, kteří využijí služby dlouhodoběji, závisí potom jen na intenzitě využívání, podle toho lze zvolit některý z připravených plánů.[39]

2.3.4 Shrnutí

Jak již bylo zmíněno, Junosphere cílí především na velké podniky zabývající se architekturou počítačových sítí. Využijí ho především zkušení profesionálové v oboru, nebo společnosti, které si chtějí předem nezávisle vyzkoušet možné budoucí řešení fyzické topologie počítačové sítě.

Velkou výhodou je zaručená stabilita, vysoká dostupnost a schopnost poskytnout nekompromisní výkon srovnatelný s fyzickým řešením. Dalším přínosem jsou také pokročilé možnosti integrace do současné síťové topologie. I přes to, že se jedná o placenou službu, lze díky ní ušetřit nemalé náklady, které by bylo nutné vynaložit na nákup velice drahých síťových zařízení od firmy Juniper. Nevýhodou je především závislost na technologiích společnosti Juniper.

3 FRAMEWORK NFV

Tato kapitola je věnovaná virtualizačním technologiím využívaným v cloudových datových centrech pro produkční účely. Primárně je zaměřená na technologie používané ve virtuální síťové infrastruktuře těchto prostředí, na její vytváření, správu a dohled. Je zde představen framework Network Fuctions Virtualization (NFV), který sjednocuje jednotlivé části cloudové infrastruktury.

3.1 Architektura NFV

NFV je koncept architektury počítačových sítí, kde jsou tradiční fyzické síťové prvky nahrazovány virtuálními stroji vykonávajícími stejnou funkcionalitu, které běží na standardních, vysoce výkonných průmyslových serverech.[40] [41]

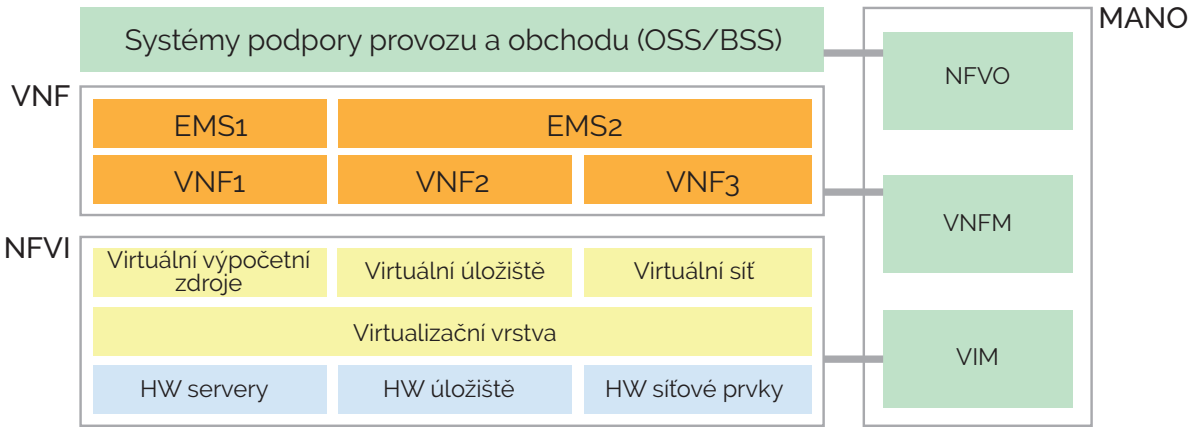
Současné počítačové sítě v podnicích a u telekomunikačních poskytovatelů se nejčastěji skládají z fyzických proprietárních zařízení. Proprietárními zařízeními se rozumí, že jejich výrobce vlastní softwarovou technologii, která je důležitá pro jejich provoz a správu. [40] [42]

Tento fakt vede k problémům s kompatibilitou při použití zařízení od různých výrobců, dále k nutnosti používat různé administrační nástroje pro různá síťová zařízení a často také k tzv. vendor lock-in. Stav vendor lock-in znamená, že v případě nutnosti rozšíření síťové infrastruktury je nutné z důvodu kompatibility pořídit znovu řešení od výrobce, od kterého pochází síťové prvky současné infrastruktury, i přesto, že by náklady na podobné řešení od konkurenčního výrobce mohly být nižší.[40] [42]

Použitím virtualizovaných síťových zařízení se lze podobným problémům vyhnout, protože softwarová řešení lze jednoduše vyvíjet jako otevřená neboli open-source. Tato otevřená řešení v oblasti virtualizace síťových zařízení velmi často vznikají právě kvůli vyhnutí se problémům s vendor lock-in. Virtualizovaná řešení jsou také už z podstaty virtualizace nezávislá na použitém hardwaru, což s sebou přináší nezávislost na výrobci fyzických zařízení.[40] [41]

The European Telecommunications Standards Institute (ETSI) je mezinárodní společenství poskytovatelů síťových služeb, kteří spolupracují na vytváření nových standardů a řešení v oblasti počítačových sítí. Tato organizace v roce 2013 představila framework NFV, který standardizuje používání konceptu NFV. Schéma tohoto frameworku je ilustrováno na obrázku č. 3.1, skládá se ze tří hlavních komponent.[43]

- Virtual Network Functions (VNF):
  VNF je softwarová implementace určité funkce síťového zařízení, která je spuštěna v jednom nebo více virtuálních strojích nebo kontejnerech. V celé infrastruktuře NFV se jich typicky vyskytuje několik.[41]
- Network Functions Virtualization Infrastructure (NFVI):
  Jedná se o subsystém, který se skládá z veškerých použitých fyzických zařízení, jako jsou například servery, standardní přepínače nebo velkokapacitní uložiště, a softwarových technologií, které umožňují virtualizaci, typicky hypervizorů.[41]
- Management and Orchestration (MANO):
  MANO je subsystém, který poskytuje administraci celého frameworku. O správu VNF se stará Virtual Networks Functions Manager (VNFM), virtualizační infrastrukturu zajišťuje Virtualized Infrastructure Manager (VIM) a o orchestraci a automatizaci celé infrastruktury NFV se stará Network Functions Virtualization Orchestrator (NFVO).[41]



Obrázek 3.1: Architektura frameworku NFV.

Zdroj: Vlastní (Zpracováno dle: NFV For Dummies [41])

Na obrázku 3.1 je také zobrazena část s názvem OSS/BSS, jedná se o systémy podpory provozu neboli Operations Support Systtems a systémy podpory obchodu Business Support Systems. Jedná se o informační systémy podniku či jiné instituce, kde je virtuální síťová infrastruktura provozována. Napojení těchto systémů na systémy v komponentě MANO zajistí integraci do aktuálního provozního informačního prostředí. [44]

Společenství poskytovatelů ETSI spolu s vydáním frameworku NFV také stanovilo některé cíle, které by měl dosáhnout.

- Rychlý rozvoj síťových a end-to-end služeb za pomoci softwarových nástrojů pro jejich nasazení a správu,
- zefektivnění provozu díky automatizaci a standardním provozním postupům,
- snížení spotřeby energie díky rovnoměrnému rozdělování zátěže a automatickému vypínání právě nepoužívaných fyzických zařízení,
- zvýšení flexibility díky přiřazování VNF konkrétnímu hardwaru,
- snížení pořizovacích a provozních nákladů.[43]

3.2 VNF

V tradičních počítačových sítích jsou síťové funkce implementovány v rámci každého fyzického zařízení jako proprietární software. NFV tento přístup mění, zde jsou síťová zařízení implementována jako virtuální stroje nebo kontejnery a síťové funkce jako software nezávislý na použitém hardwaru. Síťovou funkcí se rozumí například směrování, firewall, rozdělování zátěže paketů (tzv. load-balancing) atd.[41]

Implementace virtuálních síťových funkcí s sebou kromě nezávislosti na hardwarové platformě a výrobci také přináší možnost rozdělit určitou síťovou funkci, z nichž každá může být provozována jako samostatné VNF. Například různé funkce směrovače mohou být rozděleny do více VNF a společně pak tvořit jeden směrovač.[45]

Virtualizace síťových funkcí s sebou přináší také výhodu snadné nahraditelnosti či rozšiřitelnosti při zachování stejného hardware. Pokud například určitý druh VNF nesplní očekávané požadavky, dá se nahradit daleko snadněji než fyzická zařízení. Rozšíření lze provádět pouhou instalací nových VNF. Pokud by výkon hardwaru přestal být

dostatečný, je možnost rozšířit také ten a zachovat stejnou virtuální infrastrukturu. V případě nutnosti výměny všech nebo velké části hardwaru, na kterém je provozována virtualizační vrstva, kvůli nedostatku výkonu, je možné bez větších obtíží virtuální stroje přenést na nový hardware a opět celou infrastrukturu zachovat.[40] [42]

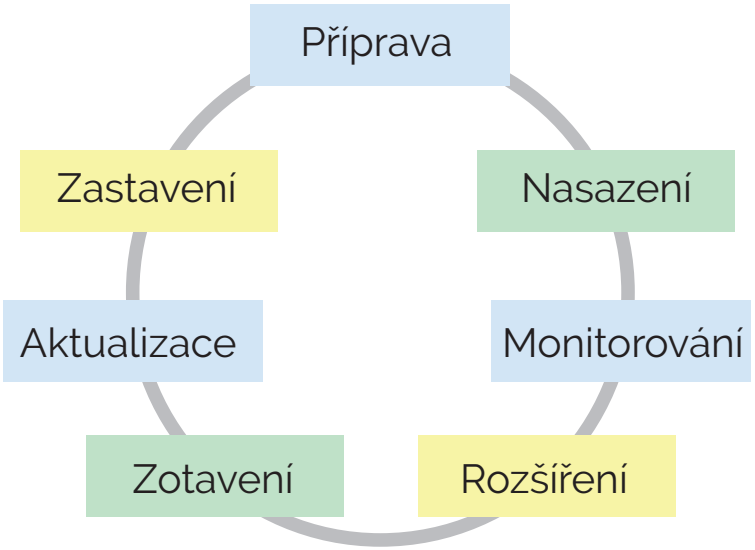
Další významnou výhodou použití NFV je lepší hospodaření s dostupnými zdroji. Fyzická síťová zařízení disponují pouze vlastními zdroji, proto je vždy třeba je dostatečně naddimenzovat. Jen zřídka se stává, že je celá síť vytížena na maximum, většinou se v určité době špičkové vytížení týká jen některých jejích částí. Z tohoto důvodu je často potřeba mít výkonný hardware v celé síti nebo její velké části, aby tyto výkyvy zvládala. To vede většinou k malému vytížení fyzických zařízení, a tedy k plýtvání zdroji, které většinu času nejsou využívané.[40]

Jelikož VNF mezi sebou sdílí hardwarové zdroje, lze s nimi hospodařit daleko efektivněji. Hardware lze nadimenzovat tak, aby zvládl špičkové vytížení pro určitou část sítě. Tyto prostředky poté mohou být dynamicky přidělovány a odebírány různým VNF na základě aktuální potřeby, a následně tak lze dosáhnout efektivnějšího hospodaření s nimi. O část těchto úkonů se mohou starat tzv. Element Management Systems (EMS). Jedná se o služby podobné VNF, které pomáhají orchestrátoru se správou VNF. Vztah EMS a VNF může být jak jedna k jedné či jedna k více.[40] [41]

U VNF je také možnost spravovat jejich životní cyklus. To znamená, že každé VNF se může nacházet v nějakém z následujících stavů:

- Příprava: V tomto stavu je VNF zadefinované ve virtualizační platformě, ale ještě není spuštěno. V praxi to znamená, že je nahrán obraz disku virtuálního stroje, nebo je připravena šablona, podle které lze VNF nasadit do provozu.[46]
- Nasazení: Jak již samotný název vypovídá, jde o spuštění daného VNF a jeho úvodní konfiguraci.[46]
- Monitorování: V tomto stavu je VNF již nasazeno a vykonává svoji funkci. Průběžně je sledován jeho stav, zda se nenaskytly nějaké potíže.[46]
- Škálování: V různých časech se může vytížení VNF měnit, proto je umožněno její škálování. Při nízkém vytížení jsou jí prostředky ubírány a při vysokém zase přidávány.[46]

- Zotavení: Pří výskytu potíží je možné na VNF aplikovat některé předem připravené postupy pro jeho uvedení do normálního provozu. Tyto kroky jsou prováděny právě ve fázi zotavení.[46]
- Aktualizace: Při možnosti instalace nových aktualizací pro VNF je možné je pomocí tohoto stavu aplikovat, přičemž není nutné smazání VNF a jeho opětovné nasazení.[46]
- Zastavení: V případě nutnosti výměny VNF, nebo její nadbytečnosti ho lze po aplikaci tohoto stavu ukončit.[46]



Obrázek 3.2: Životní cyklus VNF

Zdroj: Vlastní (Zpracováno dle: Virtual Network Functions Life Cycle Management [46])

3.3 MANO

Rozsáhlý systém virtuální síťové infrastruktury a široká škála různorodých aplikací a služeb, které mohou být součástí NFV, vyžaduje jejich jednotnou správu a orchestraci, o kterou se stará právě komponenta MANO.[41]

3.3.1 Vrstvy komponenty MANO

Komponentu MANO lze rozdělit na tři vrstvy, z nichž každá má na starosti správu jiné části frameworku. V konkrétní implementaci však jedna z aplikací, které reprezentují komponentu MANO, může spravovat i více vrstev.[41]

VIM

The Virtualized Infrastructure Manager je vrstva, která má pod správou výpočetní zdroje, úložiště a síťové zdroje. V celé infrastruktuře může být implementováno několik instancí VIM.[41] VIM plní především tyto dva hlavní úkoly:

- Správa zdrojů:
  Ta zahrnuje správu repozitářů softwaru a hypervizorů, dynamickou alokaci výpočetních, paměťových a síťových zdrojů a řízení spotřeby energie.[41]
- Správa úkonů:
  Zahrnuje úkony spojené se sběrem dat o chybách, výkonu a využití paměťových zdrojů a jejich následnou analýzu, jejíž výsledky jsou využívané pro optimalizaci výkonu.[41]

VNFM

Virtual Network Functions Manager je entita spravující jednotlivé NFV. Převážně se zaměřuje na zotavení po poruchách, správu konfigurace, účtování, řízení výkonu a zajištění bezpečnosti. Souhrn těchto odpovědností se často označuje zkratkou FCAPS (Fault, Configuration, Accounting, Performance and Security Management). VNFM je také odpovědné za vykonávání činností spojených se správou životního cyklu VNF.[41]

NFVO

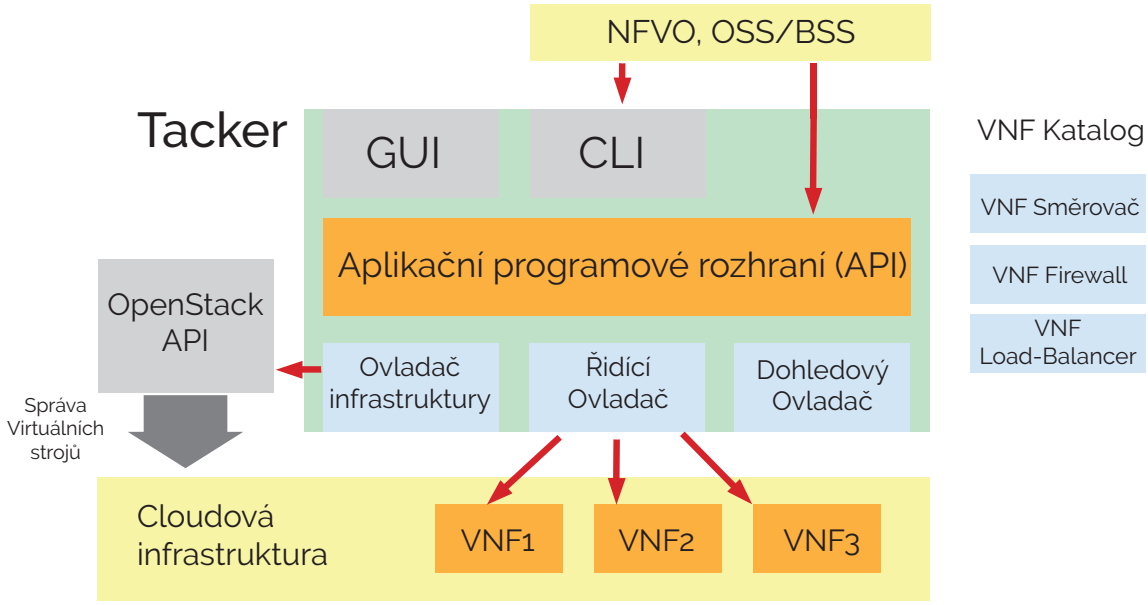
Zatímco VNF Manager se stará o správu jednotlivých VNF, NFV Orchestrator má na starosti správu služeb, které se týkají většího množství VNF. Pomocí NFVO lze vytvářet služby typu end-to-end, které využívají většího množství VNF.[41]

Jelikož NFVO poskytuje velmi rozsáhlou a často i úplnou správu infrastruktury, je nutné, aby ovládal systémy, které mají pod správou menší části infrastruktury. Pokud je například od orchestrátoru požadováno spuštění velkého množství VNF, jsou jím pouze postupně zavolány funkce VIM a VNFM, které vykonají konkrétní kroky nutné ke spuštění těchto VNF, a následně je spustí.[41]

NFVO může kromě součástí týkajících se virtuální síťové infrastruktury spravovat také jiné systémy, jako například instance virtuálních serverů, datová uložiště nebo zálohování. Orchestrátor také často poskytuje aplikační programové rozhraní, pomocí kterého může být ovládán ze systémů OSS a BSS.[41]

3.3.2 Tacker

Tacker je oficiální projekt cloudové virtualizační platformy OpenStack, který zajišťuje funkce VNFM a NFVO. Je vyvíjen jako software s otevřeným zdrojovým kódem a je navržen především pro Opentack, nicméně může být použit i v jiných podobných platformách. Řídí se standardy organizace ETSI pro tvorbu softwaru představující komponentu MANO ve frameworku NFV.[47] Nejlépe lze fungování tackeru pochopit z náčrtu jeho architektury zobrazené na obrázku č. 3.3



Obrázek 3.3: Architektura programu Tacker.

Zdroj: Vlastní (Zpracováno dle:Tacker Wiki [47])

Jednotlivé VNF vytváří Tacker na základě jejich předpisů, definovaných podle standardu TOSCA (Topology and Orchestration Specification for Cloud Applications) od společnosti Oasis. TOSCA je otevřený jazyk popisující cloudové služby pomocí šablon a plánů. Šablona popisuje strukturu dané služby, zatímco plán definuje procesy vykonávané v jednotlivých fázích jejího životního cyklu. TOSCA je velice snadno rozšiřitelný jazyk. který vývojářům umožňuje implementaci mechanismů specifických pro daný

případ užití. Pro zápis je používán formát YAML, který je velmi dobře čitelný jak programově, tak i pro člověka.[48]

Předpisy pro jednotlivé typy VNF jsou uloženy v tzv. katalogu Virtual Network Function Definitions (VNFD), kam se dostanou po provedení první fáze jejich životního cyklu, tedy přípravy.[49]

Tacker obsahuje tři základní ovladače (drivers), které vykonávají konkrétní funkce související se správou VNF. Patří mezi ně:

- Řídicí ovladač (Management Driver) vykonává akce pro správu jednotlivých instancí VNF buď napřímo, nebo častěji pomocí volání programových rozhraní řídicích modulů konkrétních typů VNF.[49]
- Ovladač infrastruktury (Infra Driver) komunikuje přes programové rozhraní se systémy, které umožňují vytváření a správu celých virtuálních infrastruktur, v současnosti se sem řadí pouze modul Heat cloudové platformy OpenStack.[49]
- Dohledový ovladač (Monitoring Driver) průběžně sleduje stav jednotlivých spuštěných VNF a v případe potíží na ně upozorní a informace o nich uloží do databáze. Navíc také může provést pokus o zotavení chybného VNF.[49]

Tacker je možné ovládat například pomocí aplikačního programového rozhraní (API), nebo příkazového řádku. Těchto dvou možností využívají především aplikace jiných orchestrátorů či aplikace OSS a BSS. Kromě nich Tacker také nabízí integraci do grafického rozhraní OpenStacku.[49]

Shrnutí

Výhodou Tackeru pro uživatele OpenStacku je to, že je vyvíjen společně s ním, tudíž je s ním plně kompatibilní a snadno se do něj integruje. Nevýhodou může být jeho orientace především na správu VNF, pro orchestraci ostatních aplikací je třeba provozovat ještě další systémy. V případě použití jiné platformy než OpenStack může být jeho integrace do ní složitější.

3.3.3 Cloudify

Cloudify je otevřeným nástrojem vyvíjeným společností Gigaspaces, který může sloužit jako NFVO a VNFM, kromě toho poskytuje také možnost orchestrace celého cloudového

prostředí. Stejně jako Tacker, nabízí monitorování prostředí a možnost automatické nápravy některých chyb.[50]

Šablony VNF se zde nazývají blueprints, jejich formát stejně jako u Tackeru splňuje specifikace TOSCA YAML. Oproti Tackeru jsou ale rozsáhlejší a nedělí se na dvě části. V jednom souboru blueprintu mohou být specifikovány všechny části aplikace, jako například její struktura, napojení na middleware, zdrojové kódy a skripty, konfigurace, logy a metriky. Blueprint může také obsahovat definice jednotlivých fází životního cyklu aplikace, kde pro každou z nich lze definovat konkrétní úkony včetně aplikací, které je budou vykonávat.[50]

Veškeré úkony týkající se správy životního cyklu pro VNF jsou prováděny pomocí modulů psaných v programovacím jazyce Python. Bez těchto modulů nemůže Cloudify vykonávat žádné operace nad prostředím, proto jsou některé moduly pro nejrozšířenější služby, jako je například OpenStack, AWS, Azure nebo Docker, nabízené společně s ním.[50]

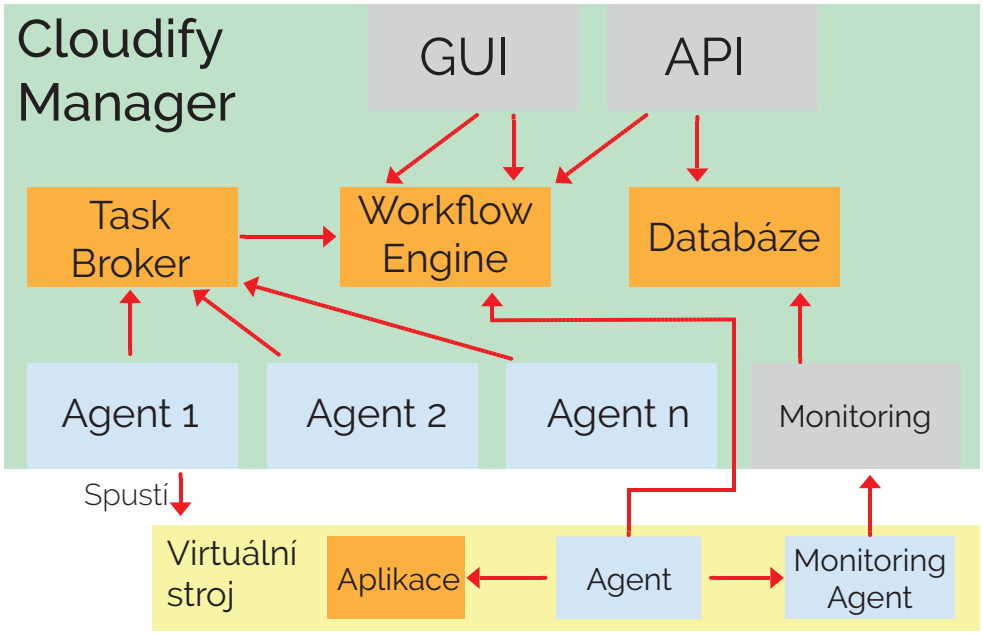
Pro specifické účely je možné tyto základní moduly libovolně upravit, nebo je nahradit vlastními. Samozřejmostí je možnost jednoduše přidat libovolné množství dalších vlastních modulů, které budou rozšiřovat kompatibilitu s dalšími systémy.

Architektura

Obrázek č. 3.4 znázorňuje high-level architekturu Cloudify Manageru. Externí aplikace a systémy OSS a BSS k němu mohou přistupovat pomocí aplikačního rozhraní REST nebo příkazového řádku. Cloudify také poskytuje vlastní grafické webové rozhraní. Vykonávání úloh zajišťuje součást Workflow Engine.[50]

Databáze se stará o uchovávání interních dat, blueprintů a metrik z dohledového systému. Pro každý plugin je vytvořen tzv. agent, který se stará o konfiguraci spravovaných aplikací a služeb. V rámci spravovaného systému mohou také běžet agenti, přičemž běžný agent může sám vykonávat některé úkony definované blueprintem přímo na spravovaném systému. Monitoring Agent informuje Cloudify o svém stavu, posílá metriky a hlásí případné chyby.[50]

Veškerou komunikaci mezi interními komponentami Cloudify Manageru zajišťuje fronta správ RabbitMQ, jejímž prostřednictvím jsou také posílány metriky a zprávy týkající se dohledu z Monitoring Agentů.[50]



Obrázek 3.4: Architektura orchestrátoru Cloudify.

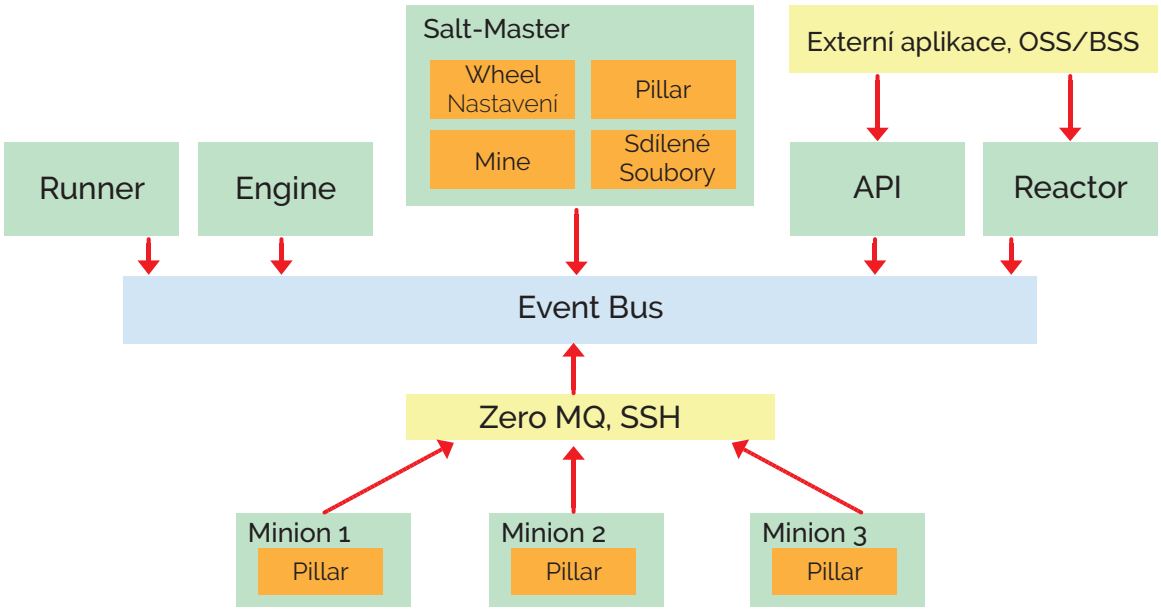
Zdroj: Vlastní (Zpracováno dle: Cloudify Documentation [50])

Shrnutí

Cloudify je silným nástrojem pro správu nejen architektury NFV, ale i celého cloudového prostředí. Je velmi flexibilní díky snadno přizpůsobitelným modulům a standardizovaným blueprintům. Nevýhodou pro snadnost nasazení je fakt, že základní moduly nejsou příliš rozsáhlé a velmi často je třeba je rozšířit. Jejich počet také není velký, a proto je také téměř vždy nutné přidat vlastní moduly, aby bylo možné pokrýt všechny součásti prostředí.

3.3.4 SaltStack

SaltStack, často nazývaný pouze Salt, je otevřený nástroj pro správu konfigurací a vzdáleného vykonávání akcí. Salt nebyl primárně navržen jako orchestrátor infrastruktury NFV, jako je tomu u Tackeru a Cloudify, ale daleko obecněji pro správu téměř jakéhokoliv systému s možností velmi velké škálovatelnosti.[51] Díky tomu se jeho architektura výrazně liší, jak lze vidět na obrázku č. 3.5.



Obrázek 3.5: Architektura orchestrátoru SaltStack.

Zdroj: Vlastní (Zpracováno dle: yet.org [52])

Salt využívá komunikační model typu server-agent, v němž komunikace probíhá tak, že server vyšle požadavek na vykonání akce jednomu či více agentům, kteří tyto akce vykonají a poté serveru vrátí jejich výsledky. Server je v tomto případě stanice, na níž běží služba s názvem Salt-Master, která řídí veškeré vykonávání a konfiguraci. Agenti běžící v rámci spravovaných stanic se nazývají Salt-Minion.[51] [52]

Další součástí Saltu jsou tzv. Runners, které zajišťují vykonávání akci na salt-masterovi. Dále je zde Reactor, což je součást, která sleduje, zda nenastaly určené typy událostí, na které může reagovat například vykonáním určitých akcí, spuštěním stavů či jiným specifickým chováním. Salt také může poskytovat aplikační rozhraní, pomocí kterého lze ovládat salt-mastera z externích systémů.[51] [52]

Pro komunikaci mezi všemi součástmi Saltu se využívá sdílená sběrnice událostí nazývaná Event Bus. Tyto události mohou být volání akcí Runneru, volání aplikačního rozhraní či události určené pro Reactor. Salt při komunikaci mezi masterem a miniony klade důraz na rychlost a bezpečnost, komunikace probíhá přes vysoce výkonnou frontu zpráv ZeroMQ, šifrování probíhá pomocí protokolu AES a autentizace je zajištěna veřejnými klíči.[51] [52]

Pro vzdálené vykonávání akcí, jako je například instalace softwarových balíčků nebo spouštění služeb, Salt nepotřebuje žádné soubory s předpisy. Tyto soubory jsou vyžadovány v případě správy konfigurace při volání tzv. stavů. Stav je v případě Saltu posloupnost vykonání určitých akcí a vynucení požadované konfigurace. Je popsán tzv. formulí, jež je předpisem těchto akcí a nutné konfigurace, přičemž má syntaxi jazyka YAML.[51] [52]

Tyto formule mohou být na rozdíl od šablon Tackeru a blueprintů Cloudify velmi obecné, a tedy použitelné pro různé typy konfigurace. Například pro vytvoření virtuálního směrovače pomocí Tackeru je třeba mít několik šablon pro různé typy jeho operačního systému. V případě použití Saltu stačí pouze jedna formule, která má typ operačního systému a případně další atributy parametrizované.[51] [52]

Tato parametrizace se provádí pomocí šablonového jazyka Jinja. Hodnoty zvolených parametrů se nahradí proměnnými, jejichž hodnoty budou nastaveny z tzv. Pillaru, předaného do stavu při jeho zavolání. Pillar je struktura metadat ve formátu YAML, která slouží pro specifikaci parametrů dosazovaných do formulí. To znamená, že při použití různých pillarů na totožnou formuli budou stavem vykonány jiné akce a konfigurace.[51] [52]

Jazyk Jinja má také velkou výhodu, že i přes to, že se nejedná o programovací jazyk, lze pomocí něj vytvářet jednoduchá rozhodováni nebo cykly. To přináší velkou flexibilitu formulí, lze je tak vytvářet daleko univerzálnější a kratší. Například díky cyklům lze efektivně spravovat opakující se části konfigurací, ve formuli je uvnitř něj definován pouze parametrizovaný blok konfigurace a v pillaru lze poté definovat libovolný počet jejích prvků.[51] [52] Příklady 3.6 a 3.7 poskytnou lepší přehled o této problematice.

Na první pohled by se mohlo zdát, že nasazení Saltu je komplikované a vyžaduje vytvoření všech modulů, stavů, formulí a pillarů. Opak je ale pravdou, komunita vyvíjející Salt se intenzivně snaží o to, aby všechny potřebné součásti potřebné pro provoz cloudové infrastruktury byly implementované již v základu. Většina funkcí, například pro vytváření virtuálních strojů či síťových zařízení, správu aplikací, jako jsou webové či souborové servery nebo databáze, je již připravena pro okamžité použití. Ve většině případů stačí pouze definovat pillary s požadovanými hodnotami a zavolat příslušné moduly či stavy.[51] [52]

```
identity:
 router:
  inet1-router:
   tenant: demo
   admin_state_up: True
   gateway_network: inet
   interfaces:
   - inet1-subnet1
   - inet1-subnet2
 router:
  inet2-router:
  tenant: demo
  admin_state_up: True
  gateway_network: inet2
  ...
```

Příklad 3.6: Pillar

```
{%- if identity.router is defined %}
{%- for router in identity.router.iteritems() %}
neutron_openstack_router_{{ router }}:
 neutronng.router_present:
  - name: {{ router_name }}
  - interfaces: {{ router.interfaces }}
  - gateway_network: {{ router.gateway_network }}
  - profile: {{ identity_name }}
  - tenant: {{ router.tenant }}
  - admin_state_up: {{ router.admin_state_up }}
  {%- if identity.endpoint_type is defined %}
  - endpoint_type: {{ identity.endpoint_type }}
  {%- endif %}
{%- endfor %}
{%- endif %}
```

Příklad 3.7: Formula

Moduly a stavy je možné snadno rozšířit, nebo naprogramovat vlastní. Všechny jsou velmi přehledně napsány v jazyce Python. Nové moduly lze ihned po vytvoření začít používat, v případě nových stavů je třeba k nim vytvořit také formule.[51] [52]

Každý minion o sobě uchovává statické informace nazývané Grains, ty obsahují například informace o hardwaru, operačním systému, síti atd. Tyto informace je možné z minionů získat a pomocí nich určit, na kterých z nich budou vykonány akce nebo aplikovány stavy. Takto lze zavolat moduly či stavy například pouze na minionech s operačním systémem Linux v Distribuci Ubuntu. Do grains lze také uložit role miniona, podle kterých se často při volání filtruje. Příklady rolí mohou být: webserver, fileserver, database atd.[51] [52]

Další informace o minionech jsou uloženy také na salt-masterovi v tzv. Salt Mine. Ten obsahuje informace nestatického charakteru, jako je například aktuální nastavení služeb. Tyto jsou obnovovány v pravidelných intervalech, aby byly udržovány aktuální.[51] [52]

Salt a NFV

I přes to, že Salt nebyl původně navržen pro NFV, lze ho snadno využít jako systém VNFM a NFVO. Jendotlivé fáze životních cyklů VNF lze navrhnout jako stavy. Oproti Tackeru a Cloudify bohužel nativně neposkytuje automatickou nápravu chyb. Této funkcionality se ovšem dá dosáhnout využitím Reactoru a vlastních modulů a stavů.

Programové rozhraní mastera poskytuje také možnost ho ovládat systémy OSS a BSS, čímž lze zajistit plnou integraci do frameworku NFV.[51] [52]

Shrnutí

Salt je velmi silným orchestračním nástrojem nejen pro NFV, ale pro celé prostředí datacentra. Lze pomocí něj nainstalovat a nakonfigurovat téměř veškerý software pro provoz cloudu. Velkou výhodou je jeho velká škálovatelnost, v některých produkčních prostředích spravuje desítky tisíc minionů najednou. Toho dosáhne pomocí tzv. Multi-Master provozu, kde se o miniony stará najednou více masterů. Kromě serverů zvládne také spravovat některá fyzická síťová zařízení, čímž tato vlastnost přináší velkou výhodu.

Nevýhodou otevřené verze Saltu může být v některých případech absence nativního grafického rozhraní, které ovšem v masivním nasazení, pro které je Salt určen, nedává moc smysl. Tuto funkci nabízí pouze jeho placená verze s názvem SaltStack Enterprise.

3.4 Infrastruktura a virtualizace

Komponenta infrastruktury a virtualizace neboli Network Functions Virtualization Infrastructure (NFVI) se skládá z fyzických zařízení, jako jsou servery, úložiště a síťové prvky, a z virtualizační vrstvy neboli hypervizoru, který je většinou součástí nějaké cloudové platformy.[41]

Skupina všech fyzických zařízení nacházejících se ve stejné lokaci, na kterých může být NFV provozované, tvoří tzv. NFVI point-of-presence (NFVI PoP). V případě provozu jednoho NFV frameworku na více geograficky rozdílných místech může existovat více NFVI PoP, každé v jedné lokaci.[41]

3.4.1 Podpůrné technologie pro NFV

Z důvodu emulace fyzických síťových portů výrazně klesá výkon zpracování síťových paketů, a tudíž i síťová propustnost síťových karet virtuálních strojů. Spolu s virtualizací síťových funkcí se rozvinuly i technologie, které zefektivňují zpracování paketů, a snižují tak propad výkonu síťové karty mezi fyzickým a virtuálním zařízením. Tyto technologie lze přirovnat k principu hardwarově asistované softwarové virtualizace.

DPDK

Data Plane Development Kit (DPDK) je soubor zdrojových kódů knihoven, určených pro rychlejší zpracování paketů na procesorech Intel řady Xeon. Společnost Intel vydala tyto knihovny jako open-source pod licencí BSD, jsou tedy volně dostupné vývojářům softwaru.[53]

Tyto knihovny si velmi rychle našly cestu do nejrozšířenějších virtuálních síťových komponent, jako je například Open vSwitch, Contrail vRouter nebo Brocade vRouter.[35] [54] [55] Intel udává, že zvýšení výkonu zpracování paketů při použití DPDK je až desetinásobně vyšší než bez něj. [56]

SR-IOV

Single-Root Input/Output Virualization (SR-IOV) je technologie od společnosti Intel, která umožňuje vytváření virtuálních síťových karet v rámci jedné fyzické na úrovni jejího hardwaru. Pro použití této funkcionality je nutné, aby ji podporoval jak hypervizor, tak síťová karta.[57]

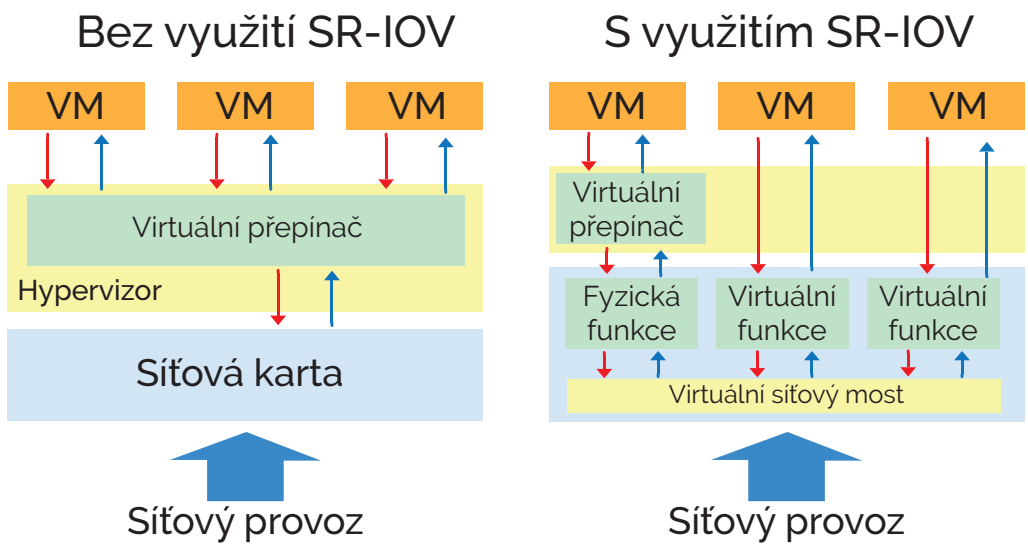
Síťová funkce v SR-IOV je oddělená sada prostředků rozhraní PCIe, především adresní prostor a bázové registry, která se pro okolí tváří jako samostatná fyzická síťová karta. Rozlišovány jsou zde dva typy síťových funkcí:

- Virtuální funkce neboli Virtual Function (VF) obsahuje pouze minimum prostředků rozhraní PCIe důležitých pro přenášení dat.[57]
- Fyzycká funkce neboli Physical Function (PF) obsahuje kompletní prostředky rozhraní PCIe, disponuje také možností ovládat veškerou funkcionalitu SR-IOV. Plní roli fyzické síťové karty.[57]

VF mohou být vytvářeny automaticky a jejich počet se může dynamicky měnit. Každá z nich může být přiřazena samostatnému fyzickému stroji. PF se vyskytuje v rámci jedné síťové karty pouze jednou a je přiřazena pouze hostujícímu operačnímu systému. [57]

SR-IOV značně snižuje režii při rozdělování paketů mezi virtuální stroje. Odpadá zde nutnost, aby hypervizor zpracovával všechny pakety, které by následně rozděloval mezi příslušné virtuální stroje.

Síťová karta může pomocí VF komunikovat s virtuálními stroji na přímo, a tím dosáhnout snížení režie a samozřejmě zvýšení propustnosti a počtu zpracovaných paketů.[57] Princip fungování SR-IOV detailněji popisuje obrázek č. 3.8



Obrázek 3.8: Architektura SR-IOV.

Zdroj: Vlastní (Zpracováno dle: PCI- SIG SR-IOV Primer [57])

3.4.2 Openstack

Jednou z možných implementací virtualizační vrstvy je OpenStack, otevřená cloudová platforma poskytující IaaS a PaaS. Jedná se o velmi oblíbený nástroj pro budování privátních cloudů. Ovšem některé společnosti, jako je například HP, eNovance nebo Rackspace, poskytují i služby veřejného cloudu postaveného na OpenStacku.[58]

OpenStack umožňuje provoz virtuálních strojů a sítí, poskytuje tzv. security groups pro tvorbu firewallových pravidel, podporuje také multitenanci, což je softwarová architektura, kdy jedna instance cloudu slouží více organizacím/divizím/oddělením atd., které jsou logicky odděleny a mají různá nastavení například oprávnění, bezpečnostních politik nebo omezení zdrojů.[59]

OpenStack se skládá z různých komponent, z nichž některé jsou nutné pro jeho provoz a ostatní jsou volitelné. Rozdělení na komponenty přináší lepší škálovatelnost, každá z nich může běžet na jednom či více fyzických serverech, jejichž počet se může měnit za provozu.

Každá komponenta je definována funkcí, kterou má plnit, a je jí přiřazena výchozí aplikace, která tuto funkci vykonává. Pro specifické účely je možné výchozí aplikaci nahradit jinou, která vykonává stejnou funkci a lépe vyhovuje požadavkům.[59]

Hlavními komponenty nutnými pro provoz OpenStacku a jejich výchozími aplikacemi jsou:

- Výpočetní - Nova:
  Nejdůležitější komponentou je výpočetní služba, která poskytuje prostředí pro provoz virtuálních strojů prostřednictvím hypervisorů. Dále nabízí zabezpečení virtuálních strojů a šifrovanou komunikaci mezi nimi.[59]

- Síťová - Neutron:
  Síťová komponenta poskytuje služby virtuální síťové infrastruktury, jako je správa sítí a podsítí, adres IP, firewallových pravidel, služeb DHCP, DNS nebo load-balancing atd. Dále umožňuje integraci různých SDN řešení, které se mohou starat o veškerou síťovou funkcionalitu a poskytnout tak lepší výkon oproti výchozímu řešení.[59]

- Autorizační - Keystone:
  Jedná se o sdílenou službu, která poskytuje autentizaci a autorizaci pro přístup k jednotlivým komponentám OpenStacku. Umožňuje také integraci různých služeb pro autentifikaci.[59]

- Přístup k obrazům disků - Glance:
  Nabízí správu obrazů disků typicky s operačními systémy či aplikacemi pro výpočetní komponentu. [59]

- Blokové úložiště - Cinder:
  Umožňuje vytvářet a spravovat diskové oddíly pro ukládání dat, které mohou být připojeny k různým instancím virtuálních strojů.[59]

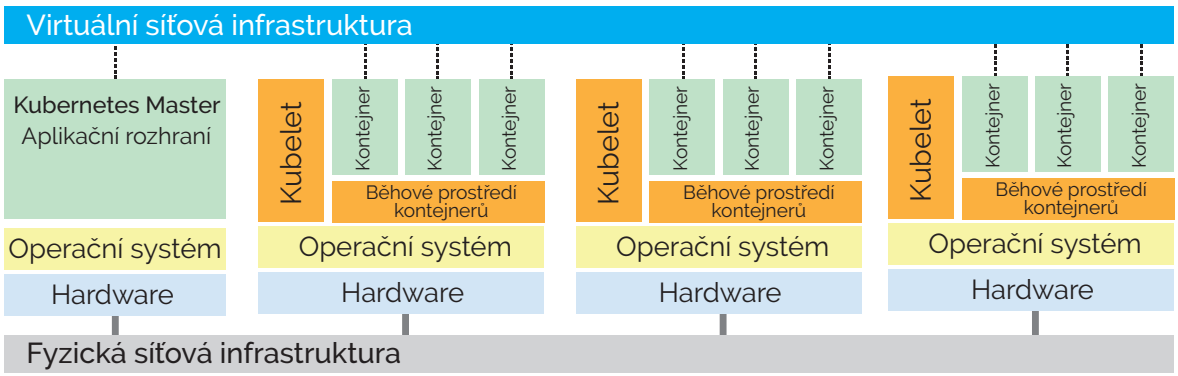
- Objektové úložiště - Swift:
  Poskytuje úložiště pro statická data, jako jsou například obrazy disků virtuálních strojů, soubory se zálohami nebo multimediální obsah.[59]

Dalšími volitelnými komponenty jsou například: webové rozhraní, databáze, fronta zpráv, orchestrace, sdílené souborové systémy, správa VNF, telemetrie, aplikační katalog a mnoho dalších.[59]

3.4.3 Kubernetes

Kubernetes je otevřená technologie vyvinutá společností Google pro automatizované nasazení, škálování a správu aplikačních kontejnerů. Jedná se o nový projekt vydaný v roce 2014, který se stal velice rychle oblíbeným, což značně podporuje jeho vývoj. Za cíle si klade: jednoduchost, rychlost, škálovatelnost a efektivitu využití hardwarových zdrojů.[60]

Kubernetes poskytuje kompletní platformu pro vybudování cloudové infrastruktury založené pouze na kontejnerech. To přináší díky výhodám kontejnerů požadovanou vysokou rychlost a flexibilitu. Architektura je velmi jednoduchá, je zobrazena na obrázku č. 3.9. Díky své jednoduchosti může Kubernetes posloužit i jako základ pro budování složitějších cloudových platforem, jako je například OpenShift, Deis nebo Eldarion.[60]



Obrázek 3.9: Architektura Kubernetes.

Zdroj: Vlastní (Zpracováno dle: Kubernetes Documentation [60])

Komponenty architektury

- Kubernetes master:

  Jedná se o hlavní řídicí uzel celé kontejnerové infrastruktury. Spravuje běh všech kontejnerů, jejich spouštění, ukončování a alokaci systémových prostředků. Poskytuje aplikační rozhraní, přes které s ním komunikují Kubelety i uživatelé, nebo vnější aplikace.[60]

- Kubelet:
  Jde o agenta, který běží na každém fyzickém serveru, kde jsou spouštěny kontejnery. Na základě příkazů od Kubernetes mastera Kubelet vykonává operace nad kontejnery, které běží na serveru pod jeho správou. Stará se také o připojování úložiště ke kontejnerům. Dále také sleduje jejich stav a získané informace předává masterovi.[60]

- Běhové prostředí kontejnerů:
  Běhové prostředí je technologie, která zajišťuje vlastní běh kontejnerů. V současnosti se využívá platforma Docker, ovšem v experimentálním vývoji je i alternativa s názvem RKT.[60]

- Pod:
  Skupina kontejnerů, které společně poskytují jednu uživatelskou službu či aplikaci, se nazývá Pod. Každý kontejner z Podu může hostovat jednu či více částí aplikace.[60]

Budoucnost pro NFV

Jelikož je Kubernetes stále relativně novou technologií, její začlenění do existujících infrastruktur nebo nahrazení současných technologií je stále v aktivním vývoji. NFV funkce nejsou vyjímkou, zde se aktivně pracuje na vytváření VNF, které mohou stabilně běžet v kontejnerech. Jedním z prvních funkčních příkladů může být jednoduchý operační systém pro virtuální směrovače s názvem Open WRT, který se již dočkal kompatibility s kontejnery.[61]

3.4.4 Shrnutí infrastruktuních řešení

V této podkapitole byly představeny zástupci obou hlavních řešení, OpenStack za hypervizory a Kubernetes jako kontejnerové řešení. Výhody hypervisorového řešení jsou především již léty ověřená stabilita a bezpečnost. Také je zde možnost široké kompatibility operačních systémů.

Navíc je již rozvinutá řada platforem, které mají podporu různých rozšíření a aplikaci. Omezení tohoto řešení je především ve ztrátě výkonu, které je způsobeno vyšší režií při provozu virtuálních strojů. Vyšší režie je i při manipulaci s virtuálními stroji, například při jejich vytváření, startu a vypínání či přesouvání.

Kontejnerová řešení odstraňují problémy s režií a tím pádem ve většině případů i s výkonem. Práce s kontejnery je velice rychlá a lze je také velice dobře škálovat oproti virtuálním strojům. Potenciál kontejnerů se začal objevovat teprve nedávno a většina technologií na nich postavených je stále ve vývoji. To s sebou přináší v určitých případech nestabilitu, problémy s bezpečností či pokles výkonu.

Například technologie Docker má v některých případech problémy s výkonem souborových vstupně-výstupních operací na uložištích připojených přes síť. V současné době existuje pouze velmi málo platforem postavených na kontejnerech, které jsou připravené pro běh v produkčním prostředí. Stále je zde také problém s kompatibilitou aplikací a operačních systémů, které se musí přizpůsobit jádru hostitelského systému.

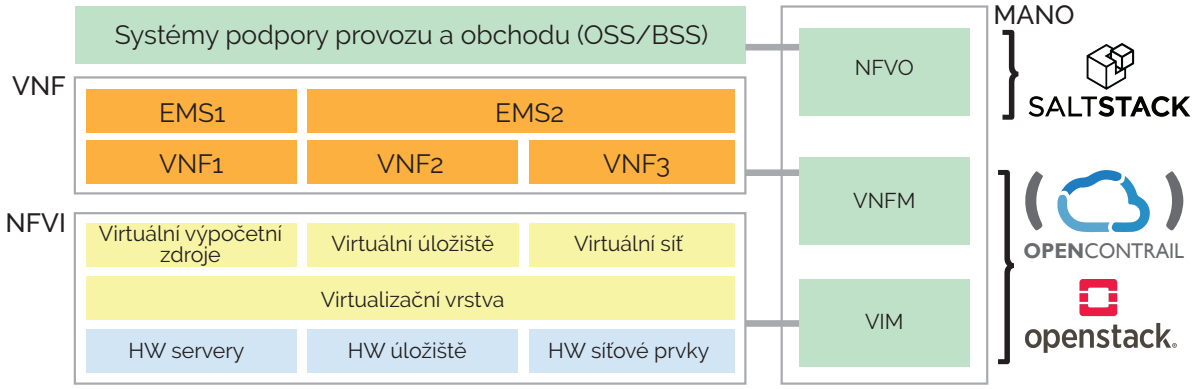
Výběr vhodného řešení tedy závisí na konkrétních případech užití, na použitých aplikacích, na míře zabezpečení a celé řadě dalších faktorů, které je třeba při výběru důkladně zvážit.

4 NFV V PRAXI

Tato kapitola obsahuje praktický příklad návrhu konkrétní implementace softwaru cloudového datového centra podle frameworku NFV. Dále je ilustrován návrh formule a pillaru pro orchestrátor Salt, umožňující spuštění tří instancí virtuálního loadbalanceru Avi Vantage od apolečnosti Avi Networks na tomto cloudu, které budou sloužit jako příklad jedné VNF.

4.1 Příklad - Implementace softwaru podle NFV

Obrázek č. 4.1 názorně ukazuje, ve kterých částech frameworku jsou jednotlivé softwarové komponenty umístěny. Jako NFVI a VIM byla zvolena cloudová platforma OpenStack s SDN rozšířením OpenContrail. Jako NFVO a VNFM posloužil nástroj SaltStack.



Obrázek 4.1: Návrh implementace NFV v cloudovém datovém centru.

Zdroj: Vlastní

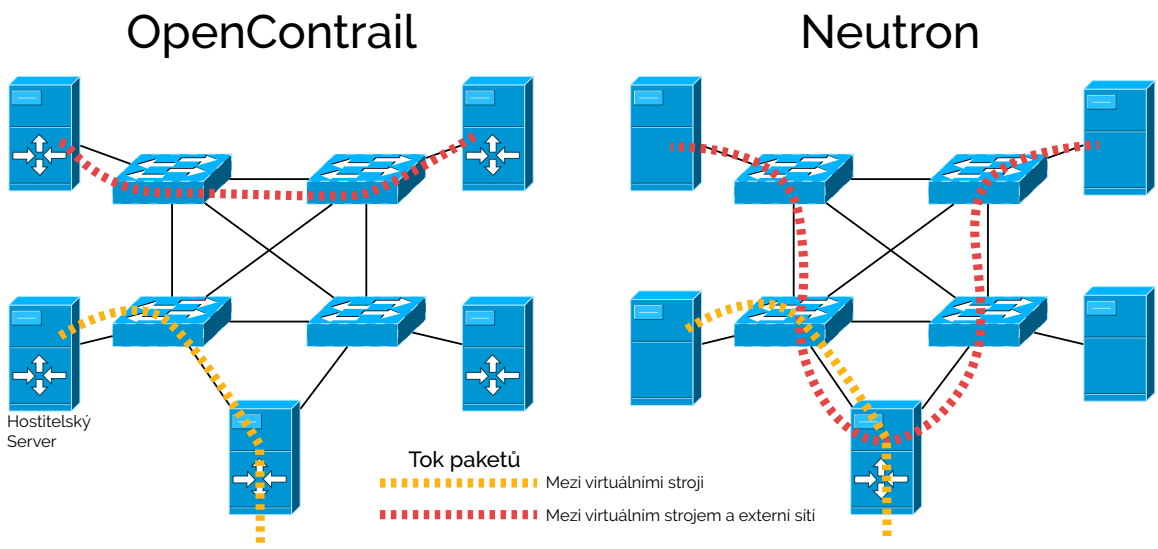
4.1.1 Vybrané komponenty architektury

OpenStack byl vybrán z důvodu jeho oblíbenosti a velkého počtu implementací v produkčních prostředích, jedná se o velmi stabilní a odladěnou platformu, a proto se pro implementaci cloudu s NFV skvěle hodí.

OpenContrail se pro použití NFV přímo nabízí, protože byl pro jeho využití již od začátku vyvíjen. Oproti standardnímu síťovému řešení v OpenStacku softwaru Neutron je OpenContrail daleko lépe škálovatelný a stabilnější hlavně při velkém počtu virtuálních strojů v řádech tisíců.[62]

Velkou výhodou oproti Neutronu je distribuovaný vRouter, to v praxi znamená, že komunikace mezi výpočetními servery, které hostují virtuální stroje, může probíhat na přímo. Tato komunikace vede při použití Neutronu přes tzv. network nodes, což jsou servery, které zajišťují veškerou komunikaci po virtuálních sítích. Tento přístup vede ke směřování veškerého síťového provozu do jednoho místa, a tedy při velkém počtu instancí k jeho zahlcování a případnému kolapsu.[62]

Použitím distribuovaného přístupu se tento problém daří odstranit, protože zátěž se rovnoměrně rozprostře mezi všechny výpočetní uzly, tím je dosaženo lepší škálovatelnosti, stability a samozřejmě vyšší síťové propustnosti. OpenContrail nabízí některé další schopnosti navíc oproti Neutronu, jako je například řetězení služeb tzv. service function chaining, load-balancing, analytické funkce či pokročilejší firewall. [63] Názornější pohled na tok paketů přes Neuron a OpenContrail poskytuje obrázek č. 4.2.



Obrázek 4.2: Rozdíl v proudu paketů mezi Neutronem a OpenContrailem.

Zdroj: Vlastní

Při použití OpenContrailu či jiné SDN technologie nedojde k úplnému odstranění Neutronu, nýbrž k vypnutí jeho funkcí pro provoz síťové infrastruktury a jejich nahrazení vybranou SDN technologií. Aplikační rozhraní Neutronu zůstává v provozu a při volání jeho funkcí jsou zavolány funkce aktuálně použité SDN technologie. Tento přístup je vhodný zejména pro zachování kompatibility s externími aplikacemi, grafickým rozhraním či orchestrátory, které aplikační rozhraní Neutronu používají.

SaltStack byl vybrán proto, že poskytuje nejširší možnosti orchestrace. Kromě správy NFV je možné pomocí něj jednoduše ovládat celé cloudové prostředí. Navíc poskytuje velmi dobrou nativní podporu funkcí OpenStacku.

4.1.2 Příprava VNF

V této fázi se předpokládá, že jsou OpenStack a OpenContrail nainstalovány a jsou plně v provozu. Postup instalace těchto technologií je zdlouhavý a nad rámec této práce. Ovšem postupy instalace jsou podrobně popsány na openstack.org a opencontrail.org.

Avi Vantage

Jako VNF byl vybrán softwarový load-balancer Avi Vantge vyvíjený společností Avi Networks. Kompletní postup jeho manuální instalace je popsán zde [64]. Cílem tohoto příkladu je nahradit tento manuální postup automatizovaným postupem pomocí Saltu.

Load-balancer se stará o rovnoměrné rozdělování zátěže, typicky požadavků od uživatelů služby, mezi všechny servery, které ji poskytují. Má tedy význam pouze v případě, že identickou službu v síti poskytují dva a více serverů.

Avi Vantage lze rozdělit na tři části:

- Service Engine (SE):
  Jedná se o virtuální stroj, který se stará o samotné rozdělování zátěže. Naslouchá na adresách IP a portech služeb, které balancuje, zpracovává požadavky od uživatelů, ty následně přesměrovává na servery, které požadovanou službu poskytují. Aby dával load-balancing smysl, je důležité mít více SE, aby nebyl veškerý síťový provoz soustředěn do jednoho místa.[65]
- Controller:
  Avi Controller je hlavním konfiguračním a řídicím místem celého systému Avi Vantage. Stará se o vytváření nových SE při vysoké zátěži a při nízkém vytížení zase o snižování jejich počtu. Typicky je provozován jako cluster tří instancí, které poskytují vysokou dostupnost.[65]
- Console:
  Tato komponenta poskytuje grafické webové rozhraní a aplikační programové rozhraní typu REST pro integraci s externími aplikacemi.[65]

Instalace umožňuje tři provozní módy. Pro tento příklad byl vybrán mód s názvem Avi-managed LBaaS mode, kde je pro Avi Controllery vytvořen speciální tenant s uživatelem s administrátorskými právy, který je schopen vytvářet AVI SE v tenantech uživatelů cloudu. Uživatelé potom mohou spravovat instance Avi SE ve svém tenantu.[64]

Postup této instalace je následující:

1. Vytvoření tenantu pro controllery.
2. Vytvoření alespoň jednoho předpisu pro instance controllerů.
3. Nahrání obrazu disku controlleru do Glance.
4. Vytvoření management sítě v Neutronu pro controllery a SE.
5. Vytvoření security-group se zadanými firewall pravidly.
6. Vytvoření instancí controlleru a přiřazení jim plovoucí adresy IP.

4.1.3 Tvorba formule a pillaru pro Salt

Požadavky jsou stanoveny, nyní přichází na řadu tvorba samotné formule, která bude odrážet výše uvedený postup. Formule pro stavy Saltu mají koncovku '.sls', v adresáři, kde jsou uloženy formule Saltu, je tedy vytvořen soubor s názvem 'avinetworks.sls'. Adresáře pro ukládání formuulí a pillarů jsou definovány v konfiguračním souboru pro službu Salt-Minion.

Na prvním kroku bude vysvětlena syntaxe kódu pro psaní formulí. Jelikož jsou pillar i formule ve formátu YAML, který pro oddělení souvisejících částí používá odsazení od kraje řádku, bude struktura popisována podle úrovní odsazení.

Nejprve je definován pillar, který představuje strukturu metadat předávaných do formule. Zavedou se do něj parametry, které se budou pro jednotlivé instance této VNF lišit. První úroveň obsahuje jednotlivé role Saltu, druhá úroveň obsahuje výčet jejích podrolí. V tomto příkladu se jedná o roli 'avinetworks', která obsahuje jednu podroli 'controller'.

Podrole již obsahuje výčet proměnných, zde se specifikuje, zda je podrole povolená (parametr 'enabled'). Parametr 'identity' specifikuje název endpointu autentifikační služby Keystone, v tomto endpointu jsou uloženy autentizační údaje, pod kterými se tento stav

bude vykonávat. Jak je již z názvu patrné, parametry 'image_location' a 'image_format' udávájí lokaci instalačního disku controlleru a jeho formát. Parametr 'public_network' specifikuje externí síť v Neutronu.

```
avinetworks:
  controller:
    enabled: true
    identity: cloud1
    image_location: http://...
    disk_format: qcow2
    public_network: INET1
```

Dříve než se začne s psaním jednotlivých kroků, je nutné definovat alespoň základní podmínku, a to zda je daná role povolena. V následující ukázce lze vidět syntaxi jazyka Jinja pro podporu rozhodování či cyklů. Dále si lze všimnout, že k parametrům z pillaru se přistupuje pomocí tečkové notace. Všechen následující kód bude umístěn uvnitř této rozhodovací podmínky.

```
{%- if server.enabled %}

{%- endif %}
```

Na první úrovni jsou definovány názvy kroků a na druhé úrovni jsou specifikovány názvy modulů v Saltu, které se při vykonávání tohoto kroku zavolají. Na třetí úrovni jsou předávány parametry pro tyto moduly. Hned u prvního parametru na třetím řádku je příklad proměnné jazyka jinja, která se umisťuje mazi dva páry složených závorek, opět se zde přes tečkovou notaci přistupuje k parametrům z pillaru.

První krok je vytvoření tenantu pro controllery. Druhým krokem je přidání uživatele admin do předtím vytvořeného tenantu. U každého volání modulu Saltu je nutné předávat parametr 'identity' kvůli autentizaci a autorizaci.

Dalším zajímavým parametrem je 'require' na 15. řádku následující ukázky. Tento parametr obsahuje seznam kroků podmiňujících vykonání aktuálního kroku. Tímto se dá značně urychlit vykonávání stavů při chybě. Pokud některý z kroků selže, poté další kroky, které na něm závisí, nebudou vůbec provedeny, protože by z důvodu závislosti selhaly také. U zbylých parametrů lze z jejich názvu snadno odvodit význam.

```
avinetworks_tenant:
  keystone.tenant_present:
    - profile: {{ server.identity }}
    - name: avinetworks
    - description: Avi Networks Vantage service project
avinetworks_user:
  keystone.user_present:
  - profile: {{ server.identity }}
  - name: admin
  - password: wouldnotreset
  - tenant: avinetworks
  - project: avinetworks
  - email: admin@admin.local
  - password_reset: False
  - require:
    - keystone: avinetworks_tenant
```

V následujících krocích je vytvořena šablona pro vytváření virtuáních strojů, která popisuje parametry virtuálního hardware, dále pak stažení obrazu instalačního disku pro budoucí instance kontroleru a jeho uložení do služby Glance.

```
avinetworks_flavor:
  novang.flavor_present:
  - name: avinetworks
  - profile: {{ server.identity }}
  - flavor_id: avinetworks
  - ram: 8192
  - disk: 160
  - vcpus: 4
  - require:
    - keystone: avinetworks_user
avinetworks_image:
  glanceng.image_present:
  - name: avinetworks
  - profile: {{ server.identity }}
  - visibility: public
  - protected: False
  - location: {{ server.image_location }}
  - disk_format: {{ server.disk_format }}
  - require:
    - keystone: avinetworks_user
```

Následující část formule obsahuje kroky týkající se vytváření síťových prostředků pomocí aplikačního rozhraní pro Neutron. Jedná se o vytvoření sítě, podsítě a routeru, který zajišťuje komunikaci s externí sítí.

```
avinetworks_network:
  neutronng.network_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - tenant: avinetworks
  - require:
    - keystone: avinetworks_user
avinetworks_subnet:
  neutronng.subnet_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - tenant: avinetworks
  - network: avinetworks
  - cidr: 10.1.0.0/24
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network
avinetworks_router:
  neutronng.router_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - tenant: avinetworks
  - interfaces:
    - avinetworks
  - gateway_network: {{ server.public_network }}
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
```

Tento krok popisuje vytvoření security-group a jejích firewallových pravidel, kterých je ve skutečnosti více, ale kvůli délce ukázky jsou vypsána až v příloze G s kompletní formulí.

```
avinetworks_secgroup:
  neutronng.security_group_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - description: AVI Networks security group
  - tenant: avinetworks
  - rules:
    - direction: egress
      ethertype: IPv6
      protocol: tcp
      remote_ip_prefix: ::/0
      port_range_min: 1
      port_range_max: 65535
```

```
      - direction: egress
        ethertype: IPv4
        remote_ip_prefix: 0.0.0.0/0
        port_range_min: 1
        port_range_max: 65535
    - require:
      - keystone: avinetworks_user
```

Nyní přichází na řadu samotné vytvoření instancí virtuálního stroje controlleru. V ukázce níže je krok pouze pro první z nich, další dvě se vytvoří stejným způsobem. Zde je třeba si dát pouze pozor na výběr vhodné adresy IP, která musí spadat do rozsahu podsítě vytvořené výše.

```
  avinetworks_instance_01:
    novang.instance_present:
    - profile: {{ server.identity }}
    - tenant_name: avinetworks
    - name: avi_ctl01
    - flavor: avinetworks
    - image: avinetworks
    - security_groups:
      - avinetworks
    - networks:
      - name: avinetworks
        v4_fixed_ip: 10.1.0.10
    - require:
      - keystone: avinetworks_user
      - novang: avinetworks_flavor
      - glanceng: avinetworks_image
      - neutronng: avinetworks_network
      - neutronng: avinetworks_subnet
      - neutronng: avinetworks_secgroup
      - file: avinetworks_init_script
```

Na závěr se těmto vytvořeným controllerům přiřadí tzv. plovoucí adresy IP, které zajišťují přístup k těmto instancím z externí sítě. Opět pro účely této práce postačí uvést příklad pouze jedné z nich.

```
avinetworks_floating_ip_01:
  neutronng.floatingip_present:
  - profile: {{ server.identity }}
  - subnet: avinetworks
  - tenant_name: avinetworks
  - name: avi_ctl01
  - network: {{ server.public_network }}
```

```
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - novang: avinetworks_instance_01
```

Jak je možné vidět na předchozích ukázkách, názvy některých modulů Saltu pro služby OpenStacku mají příponu '-ng'. To znamená, že byly původní moduly pro tyto služby rozšířeny o novou funkcionalitu a těmto rozšířeným modulům byla přidána právě tato přípona. Základní moduly v Saltu neobsahovaly některé funkce pro správu obrazů disků, tvorbu firewall pravidel a vytváření instancí. Konkrétní popis úprav v modulech je nad rámec této práce, nicméně jejich zdrojové kódy jsou obsaženy v elektronické příloze.

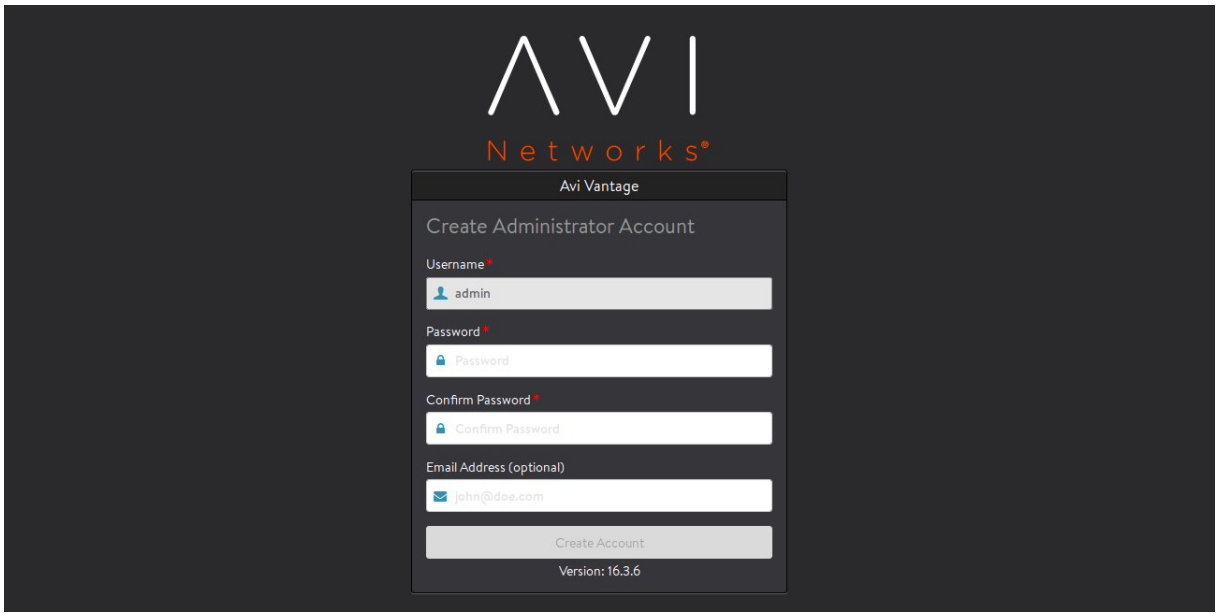
4.1.4 Spuštění a výsledky

Na závěr je třeba se ujistit, že se soubor 'controller.sls' nachází ve správném adresáři určeném pro formule Saltu, soubor pillarem je přiřazen minionovi, který bude obsahovat tuto roli, a všechny rozšířené moduly se nacházejí v adresáři pro vlastní moduly. Poté stačí na Salt-masterovi spustit příkaz pro synchronizaci všech modulů a rolí, příkaz pro synchronizaci pillaru a na konec příkaz na aplikování stavu avinetworks pro konkrétního miniona.

```
$ salt -C 'I@avinetworks:controller' saltutil.sync_all
$ salt -C 'I@avinetworks:controller' saltutil.refresh_pillar
$ salt -C 'I@avinetworks:controller' state.sls avinetworks
```

Poté Salt vypíše výsledek všech kroků, které byly provedeny. V případě některých neúspěšných je třeba z výpisu odhalit příčinu a provést změny vedoucí k nápravě. Pokud je výpis bez chyb, znamená to, že se VNF povedlo v pořádku nasadit. Všechny vytvořené součásti je možné nalézt v grafickém rozhraní Horizon nebo při výpisu pomocí příkazového řádku. Výpis o úspěšném nasazení lze nalézt v příloze H .

Nyní lze otevřít webové rozhraní některého z controllerů a ujistit se, zda jeho spuštění proběhlo bez problémů. V případě, že se zobrazí formulář pro nastavení přístupu jako na obrázku č. 4.3, je tato služba funkční a připravená k použití. Poté lze postupovat podle návodu pro přidání a nastavení služeb, mezi kterými má probíhat load-balancing.



Obrázek 4.3: Úvodní formulář pro nastavení load-balanceru Avi Vantage.

Zdroj: Vlastní

4.1.5 Shrnutí

Výsledkem tohoto příkladu je funkční formule a pillar pro orchestrátor Salt, pomocí nichž lze automatizovaně, prostřednictvím zavolání jediného příkazu, nainstalovat cluster tří instancí controlleru softwarového load-balanceru Avi Vantage včetně všech potřebných závislostí v prostředí OpenStack.

Tento příklad má také demonstrovat obecnější postup při vytváření předpisů VNF pro orchestrátory. Nezávisle na zvolené platformě, VNF či orchestrátoru, je vždy nutné se seznámit s manuální instalací vybrané VNF a poté se snažit přetransformovat tyto manuální úkony do zvoleného předpisu, ať už se jedná o šablony pro Tacker, blueprinty pro Cloudify, formule pro salt, či jakékoliv jiné předpisy pro další orchestrátory.

5 ZÁVĚR

Cílem práce bylo představit možnosti virtualizace síťových zařízení a služeb, prozkoumat nástroje pro její testování a vývoj, zhodnotit jejich výhody a nevýhody, představit framework NFV, popsat jeho architekturu, provést analýzu konkrétního softwaru, který v něm lze použít včetně porovnání a na praktickém příkladu ukázat, jak se automatizovaně nasazují virtuální síťové služby.

Začátek práce se detailně věnoval možnostem virtualizace, v průběhu práce byly také popsány podpůrné technologie, jako jsou například jmenné prostory, SDN nebo Open vSwitch. Jako nástroje pro testování a vývoj byl popsán Mininet, dále virtuální síť postavená na hypervizoru v Linuxu s pomocí Open vSwitch a na závěr profesionální služba pro virtualizaci sítě Junosphere.

V kapitole o frameworku NFV byly popsány jeho výhody oproti fyzickému síťovému řešení, dále byly rozebrány jeho komponenty. Poté byly vybráni někteří zástupci softwaru, který může sloužit pro implementaci těchto komponent, a ti byli charakterizováni, byly u nich popsány výhody a nevýhody a bylo provedeno jejich porovnání.

Praktický příklad obsahoval jednoduchý návrh konkrétní implementace privátního cloudového řešení využívajícího NFV. Pro něj byly vybrány technologie OpenStack, OpenContrail a SaltStack. Bylo zde také provedeno stručné srovnání virtuálního síťového řešení Neutron a OpenContrail.

Hlavním tématem praktického příkladu byl však postup tvorby předpisu pro orchestrátor Salt, pomocí kterého je možné nasadit virtuální síťovou službu, a to konkrétně loadbalancer Avi Vantage.

Tento předpis byl úspěšně otestován na reálném prostředí, běžícím na cloudové platformě Openstack se síťovým řešením OpenContrail. Výsledky jsou obsaženy v příloze této práce.

Závěrem lze říci, ze cíl práce byl splněn. Teoretická část obsahuje bohatý popis a porovnání technologií pro virtualizaci počítačových sítí, které jsou v současnosti používány a stále vyvíjeny. Závěrečný praktický příklad je také funkční, stejně jako dvě drobné praktické ukázky použití Mininetu a tvorby virtuální sítě na hypervizoru v Linuxu.

6 LITERATURA

[1] RUEST, Danielle a Nelson RUEST. *Virtualization: A Beginner's Guide*. New York: McGraw-Hill, 2009. ISBN 978-0-07-161402-3.

[2] HALL, Tim. A Cure for Virtual Insanity: A Vendor-Neutral Introduction to Virtualization Without the Hype. In: *ORACLE-BASE* [online]. Birmingham: ORACLE-BASE [cit. 2017-04-29]. Dostupné z: `https://oracle-base.com/articles/vm/a-cure-for-virtual-insanity`

[3] MILLER, Lawrence C. *Server Virtualization For Dummies.* 2nd Special Edition. Hoboken: John Wiley, 2013. ISBN 978-1-118-78611-6.

[4] STROBL, Marius. *Virtualization for Reliable Embedded Systems*. Norderstedt: GRIN Verlang, 2013. ISBN 978-3-656-49071-5.

[5] Mark VMI deprecated and schedule it for removal. *Linux kernel source tree* [online]. 2009 [cit. 2017-04-29]. Dostupné z: `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=d0153ca35d344d9b640dc305031b0703ba3f30f0`

[6] Support for guest OS paravirtualization using VMware VMI to be retired from new products in 2010-2011. In: *VMWare Blogs* [online]. Palo Alto, 2009 [cit. 2017-04-29]. Dostupné z: `https://blogs.vmware.com/guestosguide/2009/09/vmi-retirement.html`

[7] FISHER-OGDEN, John. *Hardware Support for Efficient Virtualization* [online]. San Diego [cit. 2017-04-29]. Dostupné z: `http://cseweb.ucsd.edu/~jfisherogden/hardwareVirt.pdf`. University of California.

[8] BROCKMEIER, Joe. Containers vs. Hypervisors: Choosing the Best Virtualization Technology. In: *Linux.com* [online]. 2010 [cit. 2017-04-29]. Dostupné z: `https://www.linux.com/news/containers-vs-hypervisors-choosing-best-virtualization-technology`

[9] NAGY, Gabor. Operating System Containers vs. Application Containers. In: *Blog RisingStack* [online]. RisingStack, 2015 [cit. 2017-04-29]. Dostupné z: https://blog.risingstack.com/operating-system-containers-vs-application-containers/

[10] CVE-2014-9357. *National Vulnerability Database* [online]. National Institute of Standards and Technology, 2014 [cit. 2017-04-29]. Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2014-9357

[11] KAMP, Poul-Henning a Robert N. M. WATSON. Jails: Confining the omnipotent r oot. In: *FreeBSD* [online]. FreeBSD, 2000 [cit. 2017-05-06]. Dostupné z: http://phk.freebsd.dk/pubs/sane2000-jail.pdf

[12] Oracle Solaris Zones Introduction. In: *Oracle Docs* [online]. Redwood City: Oracle, 2005 [cit. 2017-05-06]. Dostupné z: http://docs.oracle.com/cd/E36784_01/html/E36848/zones.intro-1.html#scrolltoc

[13] What's LXC? In: *Linux Containers* [online]. Linux Containers, 2008 [cit. 2017-05-06]. Dostupné z: https://linuxcontainers.org/lxc/introduction/

[14] BROWN, Taylor, Myles KEATING, Rick ANDERSON a Sarah COOLEY. Windows Containers. In: *Microsoft Docs* [online]. Redmond: Microsoft, 2016 [cit. 2017-05-06]. Dostupné z: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/

[15] STALLINGS, William. *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud.* Indianapolis: Pearson Education, 2016. ISBN 978-0-13-417539-3.

[16] MARSCHKE, Doug, Jeff DOYLE a Pete MOYER. *Software Defined Networking (SDN): Anatomy of OpenFlow.* Volume I. Morrisville: Lulu Publishing Services, 2015. ISBN 978-1-4834-2723-2.

[17] Namespaces. In: *Ubuntu Manpages* [online]. London: Canonical, 2016 [cit. 2017-05-06]. Dostupné z: http://manpages.ubuntu.com/manpages/xenial/man7/namespaces.7.html

[18] 800-145. *The NIST Definition of Cloud Computing*. Gaithersburg: NIST, 2011. Dostupné z: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[19] HASSAN, Qusay F. *Demystifiying Cloud Computing* [online]. Mansoura, 2011 [cit. 2017-04-29]. Dostupné z: http://static1.1.sqspcdn.com/static/f/702523/10181434/1294788395300/201101-Hassan.pdf

[20] Mininet Overview. *Mininet* [online]. Mininet Team [cit. 2017-04-29]. Dostupné z: http://mininet.org/overview/

[21] Mininet. *Github* [online]. Mininet Team [cit. 2017-04-29]. Dostupné z: https://github.com/mininet/mininet

[22] Mininet Download. *Mininet* [online]. Mininet Team [cit. 2017-04-29]. DostupnÃĺz:http://mininet.org/download/

[23] SISWANTO, Dwina Fitriyandini, Siti Amatullah a KARIMAH. Custom Mininet Topology. In: *Slideshare* [online]. LinkedIN, 2014 [cit. 2017-05-06]. Dostupné z: https://www.slideshare.net/sdnrgitb/eksperimen-custom-topology

[24] Mininet Walkthrough. *Mininet* [online]. Mininet Team [cit. 2017-04-29]. Dostupné z: http://mininet.org/walkthrough/

[25] MiniEdit App Wiki. *Github* [online]. Mininet Team [cit. 2017-04-29]. Dostupné z: https://github.com/mininet/mininet/wiki/Mini-Edit-App

[26] Network Operating System (NOS). *Techopedia* [online]. Techopedia [cit. 2017-04-29]. Dostupné z: https://www.techopedia.com/definition/3399/network-operating-system-nos

[27] Cisco Adaptive Security Virtual Appliance (ASAv). *Cisco* [online]. Cisco [cit. 2017-04-29]. Dostupné z: http://www.cisco.com/c/en/us/products/routers/cloud-services-router-1000v-series/index.html

[28] Cisco Cloud Services Router 1000V Series. *Cisco* [online]. Cisco [cit. 2017-04-29]. Dostupné z: http://www.cisco.com/c/en/us/products/security/virtual-adaptive-security-appliance-firewall/index.html

[29] vMX. *Juniper* [online]. Juniper [cit. 2017-04-29]. Dostupné z: https://www.juniper.net/us/en/products-services/routing/mx-series/vmx/

[30] vSRX Virtual Firewall. *Juniper* [online]. Juniper [cit. 2017-04-29]. Dostupné z: https://www.juniper.net/uk/en/products-services/security/srx-series/vsrx/

[31] Brocade Virtual Router. In: *Brocade* [online]. San José: Brocade [cit. 2017-04-29]. Dostupné z: http://www.brocade.com/en/products-services/software-networking/network-functions-virtualization/vrouter.html

[32] *VyOS* [online]. VyOS [cit. 2017-04-29]. Dostupné z: https://vyos.io/

[33] *OpenWRT* [online]. OpenWRT [cit. 2017-04-29]. Dostupné z: https://openwrt.org/

[34] CHIRAMMAL, Humble Devassy, Prasad MUKHEDKAR a Anil VETTATHU. *Mastering KVM Virtualization.* Birmingham: Packt Publishing, 2016. ISBN 978-1-78439-905-4.

[35] *Open vSwitch Documentation* [online]. Linux Foundation [cit. 2017-04-29]. Dostupné z: http://docs.openvswitch.org/en/latest/

[36] EMMERICH, Paul, Daniel RAUMER, Florian WOHLFART a Georg CARLE. *Performance Characteristics of Virtual Switching*. München, 2014. Technische Universität München. Dostupné z: https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/Open-vSwitch-CloudNet-14.pdf

[37] Why Open vSwitch? *Github* [online]. Linux Foundation [cit. 2017-04-29]. Dostupné z: https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst

[38] *VyOS Wiki* [online]. VyOS [cit. 2017-04-29]. Dostupné z: https://wiki.vyos.net/wiki/

[39] Junosphere Cloud Datasheet. In: *Juniper* [online]. Sunnyvale: Juniper, 2013 [cit. 2017-04-29]. Dostupné z: http://www.juniper.net/us/en/local/pdf/datasheets/1000376-en.pdf

[40] CHAYAPATHI, Rajendra. *Network function virtualization (NFV) with a touch of SDN.* Boston: Addison-Wesley, 2016. ISBN 978-0134463056.

[41] THEKKEDATH, Balamurali. *Network Functions Virtualization For Dummies.* Hoboken: Wiley, 2016. ISBN 978-1-119-14993-4.

[42] METZLER, Jim a Ashton METZLER. *The 2015 Guide to SDN and NFV* [online]. Greensboro: Webtorials, 2015 [cit. 2017-04-28]. Dostupné z: http://www.webtorials.com/main/resource/papers/webtorials/2015-Guide-to-SDN-and-NFV/2015_Guide_Complete.pdf

[43] ETSI GS NFV 001. Network Functions Virtualisation (NFV): Use Cases. Sophia Antipolis Cedex: ETSI, 2013. Dostupné z: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[44] The Definition of OSS and BSS. In: *OSS Line* [online]. OSS Line, 2010 [cit. 2017-04-29]. Dostupné z: http://www.ossline.com/2010/12/definition-oss-bss.html

[45] KHAN, Faisal. A Cheat Sheet for Understanding "NFV Architecture". In: *Telecom Lighthouse* [online]. Telecom Lighthouse [cit. 2017-04-29]. Dostupné z: http://www.telecomlighthouse.com/a-cheat-sheet-for-understanding-nfv-architecture/

[46] Virtual Network Functions Life Cycle Management. In: *Cisco* [online]. San José: Cisco [cit. 2017-04-29]. Dostupné z: http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-0/user/guide/Cisco-Elastic-Services-Controller-User-Guide-2-0/Cisco-Elastic-Services-Controller-User-Guide-2-0_chapter_0100.pdf

[47] *Tacker Wiki* [online]. OpenStack [cit. 2017-04-29]. Dostupné z: https://wiki.openstack.org/wiki/Tacker

[48] *TOSCA Simple Profile in YAML Version 1.0*. Oasis, 2016. Dostupné z: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html

[49] RAMASWAMY, Sridhar. Tacker:: VNF Lifecycle Management and Beyond. In: *IETF* [online]. IETF [cit. 2017-04-29]. Dostupné z: https://www.ietf.org/proceedings/93/slides/slides-93-nfvrg-25.pdf

[50] *Cloudify Documentation* [online]. GigaSpaces [cit. 2017-04-29]. Dostupné z: http://docs.getcloudify.org

[51] *SaltStack Docs* [online]. SaltStack [cit. 2017-05-07]. Dostupné z: https://docs.saltstack.com/en/latest/

[52] SaltStack. In: *Yet* [online]. Yet, 2014 [cit. 2017-05-07]. Dostupné z: http://www.yet.org/2016/09/salt/

[53] STANLEY, Simon. Roving Reporter: DPDK Goes Open-Source. In: *Intel* [online]. Intel, 2013 [cit. 2017-04-29]. Dostupné z: https://embedded.communities.intel.com/community/en/software/blog/2013/05/16/roving-reporter-dpdk-goes-open-source

[54] Configuring the Data Plane Development Kit (DPDK) Integrated with Contrail vRouter. In: *Juniper* [online]. Sunnyvale: Juniper, 2016 [cit. 2017-04-29]. https://www.juniper.net/techpubs/en_US/contrail3.1/topics/concept/dpdk-with-vrouter-vnc.html

[55] Brocade vRouter Architecture. In: *Brocade* [online]. San José: Brocade [cit. 2017-04-29]. Dostupné z: http://www.brocade.com/content/html/en/technical-documentation-library/5600-vrouter-41R1-library/GUID-B07BD5C1-5E8A-497C-9FE6-DA9B31D27369.html

[56] Data Plane Development Kit (DPDK). In: *Intel* [online]. Santa Clara: Intel [cit. 2017-04-29]. Dostupné z: http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html

[57] 321211-002. *PCI- SIG SR-IOV Primer.* Rev 2.5. Santa Clara: Intel, 2011. http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html

[58] FIFIELD, Tom, Diane FLEMING, Anne GENTLE, Lorin HOCHSTEIN, Jonathan PROLUX, Everett TOEWS a Joe TOPJIAN. *OpenStack Operations Guide*. Sebastopol: O'Reilly, 2014. ISBN 978-1-491-94695-4.

[59] *OpenStack Documentation* [online]. OpenStack [cit. 2017-04-29]. Dostupné z: `https://docs.openstack.org`

[60] *Kubernetes Documentation* [online]. Linux Foundation [cit. 2017-04-29]. Dostupné z: `https://kubernetes.io/docs/home/`

[61] Docker OpenWrt Image. *Wiki OpenWrt* [online]. OpenWrt [cit. 2017-04-29]. Dostupné z: `https://wiki.openwrt.org/doc/howto/docker_openwrt_image`

[62] SINGLA, Ankur a Bruno RIJSMAN. *Day One: Understanding OpenContrail Architecture*. Sunnyvale: Juniper, 2013. ISBN 978-1-936779-71-0.

[63] OpenContrail Datasheet. In: *Juniper* [online]. Sunnyvale: Juniper, 2013 [cit. 2017-04-29]. Dostupné z: `http://www.juniper.net/us/en/local/pdf/datasheets/1000472-en.pdf`

[64] Installing Avi Vantage for OpenStack. In: *Avi Network* [online]. Santa Clara: Avi Network [cit. 2017-04-29]. Dostupné z: `https://kb.avinetworks.com/installing-avi-vantage-for-openstack-16-1`

[65] Architectural Overview. In: *Avi Network* [online]. Santa Clara: Avi Network [cit. 2017-04-29]. Dostupné z: `https://kb.avinetworks.com/docs/architectural-overview/`

7 SEZNAM PŘÍLOH

Příloha A Konfigurace směrovače R1

```
interfaces {
    ethernet eth0 {
        address 192.168.0.1/24
        duplex auto
        hw-id 52:54:00:b6:f5:05
        smp_affinity auto
        speed auto
    }
    ethernet eth1 {
        address 192.168.1.1/24
        duplex auto
        hw-id 52:54:00:68:e0:6f
        smp_affinity auto
        speed auto
    }
    loopback lo {
        address 1.1.1.1/32
    }
}
protocols {
    ospf {
        area 0 {
            network 192.168.0.0/24
            network 192.168.1.0/24
        }
        parameters {
            abr-type cisco
            router-id 1.1.1.1
        }
    }
}
```

Příloha B Konfigurace směrovače R2

```
hostname R2
interface GigabitEthernet1
 ip address 192.168.0.2 255.255.255.0
 ip ospf lls disable
 ip ospf 2 area 0
 negotiation auto
interface GigabitEthernet2
 ip address 192.168.2.1 255.255.255.0
 ip ospf 2 area 0
 negotiation auto
router ospf 2
 router-id 2.2.2.2
 network 192.168.0.0 0.0.0.255 area 0
 network 192.168.2.0 0.0.0.255 area 0
```

Příloha C Výstup příkazu ping hosta H1

```
$ ping 192.168.1.6
PING 192.168.1.6 (192.168.1.6): 56 data bytes
64 bytes from 192.168.1.6: seq=0 ttl=64 time=0.414 ms
64 bytes from 192.168.1.6: seq=1 ttl=64 time=1.729 ms
64 bytes from 192.168.1.6: seq=2 ttl=64 time=0.885 ms
64 bytes from 192.168.1.6: seq=3 ttl=64 time=0.813 ms

$ ping 192.168.2.5
PING 192.168.2.5 (192.168.2.5): 56 data bytes
64 bytes from 192.168.2.5: seq=1 ttl=62 time=3.504 ms
64 bytes from 192.168.2.5: seq=2 ttl=62 time=2.664 ms
64 bytes from 192.168.2.5: seq=3 ttl=62 time=2.702 ms
64 bytes from 192.168.2.5: seq=4 ttl=62 time=2.964 ms

$ ping 192.168.2.6
PING 192.168.2.6 (192.168.2.6): 56 data bytes
64 bytes from 192.168.2.6: seq=1 ttl=62 time=4.304 ms
64 bytes from 192.168.2.6: seq=2 ttl=62 time=3.289 ms
64 bytes from 192.168.2.6: seq=3 ttl=62 time=2.861 ms
64 bytes from 192.168.2.6: seq=4 ttl=62 time=2.338 ms
```

Příloha D Výstup příkazu ping hosta H2

```
$ ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5): 56 data bytes
64 bytes from 192.168.1.5: seq=0 ttl=64 time=1.123 ms
64 bytes from 192.168.1.5: seq=1 ttl=64 time=0.719 ms
64 bytes from 192.168.1.5: seq=2 ttl=64 time=0.680 ms
64 bytes from 192.168.1.5: seq=3 ttl=64 time=0.932 ms

$ ping 192.168.2.5
PING 192.168.2.5 (192.168.2.5): 56 data bytes
64 bytes from 192.168.2.5: seq=0 ttl=62 time=2.561 ms
64 bytes from 192.168.2.5: seq=1 ttl=62 time=2.720 ms
64 bytes from 192.168.2.5: seq=2 ttl=62 time=3.187 ms
64 bytes from 192.168.2.5: seq=3 ttl=62 time=3.265 ms

$ ping 192.168.2.6
PING 192.168.2.6 (192.168.2.6): 56 data bytes
64 bytes from 192.168.2.6: seq=0 ttl=62 time=1.328 ms
64 bytes from 192.168.2.6: seq=1 ttl=62 time=1.451 ms
64 bytes from 192.168.2.6: seq=2 ttl=62 time=1.486 ms
64 bytes from 192.168.2.6: seq=3 ttl=62 time=6.045 ms
```

Příloha E Výstup příkazu ping hosta H3

```
$ ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5): 56 data bytes
64 bytes from 192.168.1.5: seq=0 ttl=62 time=2.085 ms
64 bytes from 192.168.1.5: seq=1 ttl=62 time=2.951 ms
64 bytes from 192.168.1.5: seq=2 ttl=62 time=3.051 ms
64 bytes from 192.168.1.5: seq=3 ttl=62 time=2.613 ms

$ ping 192.168.1.6
PING 192.168.1.6 (192.168.1.6): 56 data bytes
64 bytes from 192.168.1.6: seq=0 ttl=62 time=2.028 ms
64 bytes from 192.168.1.6: seq=1 ttl=62 time=2.783 ms
64 bytes from 192.168.1.6: seq=2 ttl=62 time=1.826 ms
64 bytes from 192.168.1.6: seq=3 ttl=62 time=4.176 ms

$ ping 192.168.2.6
PING 192.168.2.6 (192.168.2.6): 56 data bytes
64 bytes from 192.168.2.6: seq=0 ttl=64 time=1.935 ms
64 bytes from 192.168.2.6: seq=1 ttl=64 time=0.931 ms
64 bytes from 192.168.2.6: seq=2 ttl=64 time=0.804 ms
64 bytes from 192.168.2.6: seq=3 ttl=64 time=0.683 ms
```

Příloha F Výstup příkazu ping hosta H4

```
$ ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5): 56 data bytes
64 bytes from 192.168.1.5: seq=0 ttl=62 time=2.965 ms
64 bytes from 192.168.1.5: seq=1 ttl=62 time=2.822 ms
64 bytes from 192.168.1.5: seq=2 ttl=62 time=3.087 ms
64 bytes from 192.168.1.5: seq=3 ttl=62 time=2.920 ms

$ ping 192.168.1.6
PING 192.168.1.6 (192.168.1.6): 56 data bytes
64 bytes from 192.168.1.6: seq=0 ttl=62 time=1.997 ms
64 bytes from 192.168.1.6: seq=1 ttl=62 time=2.358 ms
64 bytes from 192.168.1.6: seq=2 ttl=62 time=2.761 ms
64 bytes from 192.168.1.6: seq=3 ttl=62 time=2.015 ms

$ ping 192.168.2.5
PING 192.168.2.5 (192.168.2.5): 56 data bytes
64 bytes from 192.168.2.5: seq=0 ttl=64 time=0.716 ms
64 bytes from 192.168.2.5: seq=1 ttl=64 time=0.525 ms
64 bytes from 192.168.2.5: seq=2 ttl=64 time=1.665 ms
64 bytes from 192.168.2.5: seq=3 ttl=64 time=0.886 ms
```

Příloha G Salt formule z příkladu pro load balancer Avi Vantage

```
{%- if server.enabled %}

  avinetworks_tenant:
    keystone.tenant_present:
      - profile: {{ server.identity }}
      - name: avinetworks
      - description: Avi Networks Vantage service project

  avinetworks_user:
    keystone.user_present:
    - profile: {{ server.identity }}
    - name: admin
    - password: wouldnotreset
    - tenant: avinetworks
    - project: avinetworks
    - email: admin@admin.local
    - password_reset: False
    - require:
      - keystone: avinetworks_tenant

  avinetworks_flavor:
    novang.flavor_present:
    - name: avinetworks
    - profile: {{ server.identity }}
```

```
  - flavor_id: avinetworks
  - ram: 8192
  - disk: 160
  - vcpus: 4
  - require:
    - keystone: avinetworks_user

avinetworks_image:
  glanceng.image_present:
  - name: avinetworks
  - profile: {{ server.identity }}
  - visibility: public
  - protected: False
  - location: {{ server.image_location }}
  - disk_format: {{ server.disk_format }}
  - require:
    - keystone: avinetworks_user

avinetworks_network:
  neutronng.network_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - tenant: avinetworks
  - require:
    - keystone: avinetworks_user

avinetworks_subnet:
  neutronng.subnet_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - tenant: avinetworks
  - network: avinetworks
  - cidr: 10.1.0.0/24
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network

avinetworks_router:
  neutronng.router_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - tenant: avinetworks
  - interfaces:
    - avinetworks
  - gateway_network: {{ server.public_network }}
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network
```

```
    - neutronng: avinetworks_subnet

avinetworks_secgroup:
  neutronng.security_group_present:
  - profile: {{ server.identity }}
  - name: avinetworks
  - description: AVI Networks security group
  - tenant: avinetworks
  - rules:
    - direction: egress
      ethertype: IPv6
      protocol: tcp
      remote_ip_prefix: ::/0
      port_range_min: 1
      port_range_max: 65535
    - direction: egress
      ethertype: IPv4
      remote_ip_prefix: 0.0.0.0/0
      port_range_min: 1
      port_range_max: 65535
    - direction: ingress
      ethertype: IPv4
      protocol: icmp
      remote_ip_prefix: 0.0.0.0/0
    - direction: ingress
      ethertype: IPv4
      protocol: tcp
      port_range_min: 22
      port_range_max: 22
      remote_ip_prefix: 0.0.0.0/0
    - direction: ingress
      ethertype: IPv4
      protocol: tcp
      port_range_min: 80
      port_range_max: 80
      remote_ip_prefix: 0.0.0.0/0
    - direction: ingress
      ethertype: IPv4
      protocol: tcp
      port_range_min: 123
      port_range_max: 123
      remote_ip_prefix: 10.1.0.0/24
    - direction: ingress
      ethertype: IPv4
      protocol: tcp
      port_range_min: 443
      port_range_max: 443
      remote_ip_prefix: 0.0.0.0/0
```

```
    - require:
      - keystone: avinetworks_user

avinetworks_init_script:
  file.managed:
  - name: /etc/avi_init.sh
  - source: salt://avinetworks/files/avi_init
  - defaults:
      saltmaster: {{ server.saltmaster_ip }}
  - template: jinja

avinetworks_instance_01:
  novang.instance_present:
  - profile: {{ server.identity }}
  - tenant_name: avinetworks
  - name: avi_ctl01
  - flavor: avinetworks
  - image: avinetworks
  - security_groups:
    - avinetworks
  - networks:
    - name: avinetworks
      v4_fixed_ip: 10.1.0.10
  - require:
    - keystone: avinetworks_user
    - novang: avinetworks_flavor
    - glanceng: avinetworks_image
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - neutronng: avinetworks_secgroup
    - file: avinetworks_init_script

avinetworks_instance_02:
  novang.instance_present:
  - profile: {{ server.identity }}
  - tenant_name: avinetworks
  - name: avi_ctl02
  - flavor: avinetworks
  - image: avinetworks
  - security_groups:
    - avinetworks
  - networks:
    - name: avinetworks
      v4_fixed_ip: 10.1.0.11
  - require:
    - keystone: avinetworks_user
    - novang: avinetworks_flavor
    - glanceng: avinetworks_image
```

```
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - neutronng: avinetworks_secgroup
    - file: avinetworks_init_script

avinetworks_instance_03:
  novang.instance_present:
  - profile: {{ server.identity }}
  - tenant_name: avinetworks
  - name: avi_ctl03
  - flavor: avinetworks
  - image: avinetworks
  - security_groups:
    - avinetworks
  - networks:
    - name: avinetworks
      v4_fixed_ip: 10.1.0.12
  - require:
    - keystone: avinetworks_user
    - novang: avinetworks_flavor
    - glanceng: avinetworks_image
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - neutronng: avinetworks_secgroup
    - file: avinetworks_init_script

avinetworks_floating_ip_01:
  neutronng.floatingip_present:
  - profile: {{ server.identity }}
  - subnet: avinetworks
  - tenant_name: avinetworks
  - name: avi_ctl01
  - network: {{ server.public_network }}
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - novang: avinetworks_instance_01

avinetworks_floating_ip_02:
  neutronng.floatingip_present:
  - profile: {{ server.identity }}
  - subnet: avinetworks
  - tenant_name: avinetworks
  - name: avi_ctl02
  - network: {{ server.public_network }}
  - require:
    - keystone: avinetworks_user
```

```
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - novang: avinetworks_instance_02

avinetworks_floating_ip_03:
  neutronng.floatingip_present:
  - profile: {{ server.identity }}
  - subnet: avinetworks
  - tenant_name: avinetworks
  - name: avi_ctl03
  - network: {{ server.public_network }}
  - require:
    - keystone: avinetworks_user
    - neutronng: avinetworks_network
    - neutronng: avinetworks_subnet
    - novang: avinetworks_instance_03

{%- endif %}
```

Příloha H Výstup ze spuštění Salt stavu 'avinetworks'

```
local:
  ----------
          ID: avinetworks_tenant
    Function: keystone.tenant_present
        Name: avinetworks
      Result: True
     Comment: Tenant "avinetworks" has been added
     Started: 11:36:23.835594
    Duration: 647.15 ms
     Changes:
              ----------
              Tenant:
                  Created
  ----------
          ID: avinetworks_user
    Function: keystone.user_present
        Name: admin
      Result: True
     Comment: User "admin" has been updated
     Started: 11:36:24.483323
    Duration: 883.327 ms
     Changes:
              ----------
              Email:
                  Updated
              Tenant:
                  Added to "avinetworks" tenant
```

```
----------
          ID: avinetworks_flavor
    Function: novang.flavor_present
        Name: avinetworks
      Result: True
     Comment: Flavor "avinetworks" has been created
     Started: 12:17:25.788855
    Duration: 576.26 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          ac42fade-86b1-4f5c-8276-acdd4861ef46
                      ram:
                          8192
                      disk:
                          160
                      vcpus:
                          4
                  old:
                      None
----------
          ID: avinetworks_image
    Function: glanceng.image_present
        Name: avinetworks
      Result: True
     Comment: "visibility" is correct (public).
              "protected" is correct (False).
     Started: 12:17:26.838367
    Duration: 7566.206 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          bf34ba97-5466-4463-a224-4459416ccfd2
                  old:
                      None
----------
          ID: avinetworks_network
    Function: neutronng.network_present
        Name: avinetworks
      Result: True
```

```
     Comment: Network "avinetworks" has been created
     Started: 12:17:27.128648
    Duration: 754.134 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          4bce29e7-0bb3-4477-88d0-275be87576a1
                  old:
                      None
----------
          ID: avinetworks_subnet
    Function: neutronng.subnet_present
        Name: avinetworks
      Result: True
     Comment: Subnet "avinetworks" has been created
     Started: 12:17:28.74634
    Duration: 345.563 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          80e17c03-7ed8-4e61-9a70-95f35a010fe5
                      cidr:
                          10.1.0.0_24
                  old:
                      None
----------
          ID: avinetworks_router
    Function: neutronng.router_present
        Name: avinetworks
      Result: True
     Comment: Router "avinetworks" has been created
     Started: 12:17:30.87362
    Duration: 2308.098 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
```

```
                        4786b24d-965f-4a8b-9caf-ac272163116f
                    gateway:
                        INET1
                old:
                    None
----------
          ID: avinetworks_secgroup
    Function: neutronng.security_group_present
        Name: avinetworks
      Result: True
     Comment: Security group "avinetworks" has been created
     Started: 12:17:32.74634
    Duration: 3859.297 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          775b6524-23e1-4331-8009-217cc87fd4ec
                      description:
                          AVI Networks security group
                  old:
                      None
----------
          ID: avinetworks_instance01
    Function: novang.instance_present
        Name: avi_ctl01
      Result: True
     Comment: Instance "avi_ctl01" has been launched
     Started: 12:17:36.74634
    Duration: 2340.476 ms
     Changes:
              ----------
              avi_ctl01:
                  ----------
                  new:
                      ----------
                      id:
                          9a56de39-4e08-4d47-81c1-9edb9111d16b
                      flavor:
                          avinetworks
                      image:
                          avinetworks
                      networks:
                          - avinetworks
                      v4_fixed_ips:
```

```
                        - 10.1.0.10
                    status:
                        spawning
                old:
                    None
----------
          ID: avinetworks_instance02
    Function: novang.instance_present
        Name: avi_ctl02
      Result: True
     Comment: Instance "avi_ctl02" has been launched
     Started: 12:17:39.17393
    Duration: 2403.846 ms
     Changes:
              ----------
              avi_ctl02:
                  ----------
                  new:
                      ----------
                      id:
                          15264957-5c4d-488e-be3b-8af9c8d01189
                      flavor:
                          avinetworks
                      image:
                          avinetworks
                      networks:
                          - avinetworks
                      v4_fixed_ips:
                          - 10.1.0.11
                      status:
                          spawning
                  old:
                      None
----------
          ID: avinetworks_instance03
    Function: novang.instance_present
        Name: avi_ctl03
      Result: True
     Comment: Instance "avi_ctl03" has been launched
     Started: 12:17:41.98765
    Duration: 2479.345 ms
     Changes:
              ----------
              avi_ctl03:
                  ----------
                  new:
                      ----------
                      id:
```

```
                        700bea7e-856f-49f4-98bb-b163e3c0a562
                    flavor:
                        avinetworks
                    image:
                        avinetworks
                    networks:
                        - avinetworks
                    v4_fixed_ips:
                        - 10.1.0.12
                    status:
                        spawning
                old:
                    None
----------
          ID: avinetworks_floating_ip_01
    Function: neutronng.floatingip_present
      Result: True
     Comment: Floating IP has been assigned
     Started: 12:17:44.50323
    Duration: 987.297 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          85227199-cac2-48c4-b874-9b27b0acd3a6
                      network:
                          INET1
                      instance:
                          avi_ctl01
                  old:
                      None
----------
          ID: avinetworks_floating_ip_02
    Function: neutronng.floatingip_present
      Result: True
     Comment: Floating IP has been assigned
     Started: 12:17:45.64326
    Duration: 1209.367 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
```

```
                        d0b2cfb1-4c96-4dbf-995c-ddb2c09b9636
                    network:
                        INET1
                    instance:
                        avi_ctl02
                old:
                    None
----------
          ID: avinetworks_floating_ip_03
    Function: neutronng.floatingip_present
      Result: True
     Comment: Floating IP has been assigned
     Started: 12:17:47.29326
    Duration: 1082.104 ms
     Changes:
              ----------
              avinetworks:
                  ----------
                  new:
                      ----------
                      id:
                          c47e5576-33f1-11e7-a919-92ebcb67fe33
                      network:
                          INET1
                      instance:
                          avi_ctl03
                  old:
                      None
```

Příloha I CD s digitální podobu DP a elektronickými přílohami

CD obsahuje diplomovou práci ve formátu PDF, výpis komunikace z testování příkladu sítě na hypervizoru KVM ze 2. kapitoly zaznamenané programem Wireshark a zdrojové kódy vlastních modulů a stavů pro Salt nutné ke zprovoznění příkladu ve 4. kapitole.