VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Dmytro Dubovyi

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH LABORATORNÍCH ÚLOH V OBLASTI PROGRAMOVATELNOSTI SÍTÍ
DESIGN OF LABORATORY EXERCISES IN THE FIELD OF NETWORK PROGRAMMABILITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Dmytro Dubovyi**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Vladislav Škorpil, CSc.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Dmytro Dubovyi ***ID:*** 201236
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Návrh laboratorních úloh v oblasti programovatelnosti sítí

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je zhodnocení současného vývoje v oblasti SDN (Software Defined Network) a možností programovatelnosti (Network Programmability) SDN prvků za pomocí rozhraní pro programování aplikací API (Application Programming Interface). Navrhněte a vytvořte laboratorní úlohy na toto výše uvedené téma. Součástí návrhu je i zařízení BIG-IP F5 Network. Uvažováno je také využití Python a Ansible. Výstupem diplomové práce je teoretická část a část praktická, která obsahuje konkrétní skripty API a návrh minimálně dvou laboratorních úloh včetně jejich zpracovaného zadání.

DOPORUČENÁ LITERATURA:

[1] Getting Started with Programmability [online]. F5 Networks, 2019 [cit. 2019-09-07]. Dostupné z: https://bit.ly/2ky6zy8

[2] PILGRIM, Mark. Ponořme se do Python(u) 3: Dive into Python 3. Praha: CZ.NIC, c2010. CZ.NIC. ISBN 978-80-904248-2-1.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: doc. Ing. Vladislav Škorpil, CSc.
Konzultant: Ing. Václav Oujezský, Ph.D. (VUT Brno)

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Cílem diplomové práce je zhodnocení současného vývoje v oblasti SDN a možnosti programovatelnosti SDN prvků pomocí rozhraní pro programování aplikací API. V teoretické části jsou popsány: základní architektura SDN, provoz uvnitř SDN mezi jeho jednotlivými vrstvami, protokoly komunikace „Southbound“ rozhraní a „Northbound“ rozhraní. Další část práce se zabývá možností programovatelnosti SDN prvků pomocí API. Poslední kapitola teoretické časti vylíčí současný vývoj v oblasti SDN. Praktická část je věnována vytvoření dvou laboratorních úloh, které se zabývají programováním SDN API. Laboratorní úlohy se týkají programování BIG-IP od společnosti F5 Network a směrovačů od společnosti Arista Network. Programování probíhá pomocí Pythonu přes REST API pro BIG-IP, nebo eAPI pro Arista EOS. Pro stejný účel byl také využít nastroj pro nastavení Ansible.

KLÍČOVÁ SLOVA

SDN, API, architektura SDN, SDN kontrolér, RESTful API, Ansible, CAGR, Python, BIG-IP, eAPI, EOS.

ABSTRACT

The aim of the graduation thesis is to evaluate the current development in the field of SDN and the possibility of programmability of SDN elements using the application programming interface. The first theoretical chapter describes the following: the basic architecture of SDN, the traffic within SDN between its individual layers, the communication protocols Southbound interface and Northbound interface. The second chapter of the thesis deals with the programmability of SDN elements with the help of API. The third theoretical chapter describes the current development in the field of SDN. The practical part of the thesis is devoted to creation of two laboratory tasks dealing with the programming of the SDN API. Laboratory tasks include BIG-IP programming from F5 Network and routers from Arista Network. Programming is done using Python via REST API for BIG-IP, or eAPI for Arista EOS. The Ansible setup tool is also used for the same purpose.

KEYWORDS

SDN, API, SDN architecture, SDN controller, RESTful API, Ansible, CAGR, Python, BIG-IP, eAPI, EOS.

DUBOVYI, Dmytro. *Síťová programmabilita - návrh a implementace laboratorních úloh*. Brno, Rok, 88 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Vladislav Škorpil, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Síťová programmabilita - návrh a implementace laboratorních úloh“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval konzultantovi diplomové práce Ing. Václavovi Oujezskému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Mé velké poděkování patří také panu doc. Ing. Vladislavu Škorpilovi, CSc. za jeho drahocenný čas, odborné rady a veškerou pomoc při zpracování teto práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Počítačové sítě jsou strategickým faktorem ve vývoji moderních IT technologií, ale síťová architektura není vždy schopna přiměřeně a účinně reagovat na nové potřeby. V moderním světě a v oblasti informačních technologií stále vidíme narůst nároků na flexibilitu a škálovatelnost počítačových sítí. Současně se síť ve své klasické podobě (ovládání pomocí příkazového řádku a konfiguračních souborů) stává omezujícím faktorem ve vývoji výpočetní infrastruktury. Tradiční sítě jsou primárně statické a neodpovídají rychlé dynamice rozvoje moderního IT podnikání. Schopnost škálování tradičních sítí nesplňuje požadavky velkých podniků a poskytovatelů služeb (Facebook, Google, Microsoft a Verizon) a správa distribuovaných zařízení tradičních sítí je příliš komplikovaná a neúčinná. Vazba na vybraného výrobce sítě nezaručuje podporu budoucích aplikací a služeb. Klasické přístupy k řešení těchto problémů, například založené na síťové virtualizaci VLAN (Virtual Local Area Network), VRF (Virtual Routing and Forwarding), neodpovídají úrovni vývoje systému virtualizace. Tyto faktory vedou k tomu, že tradiční počítačové sítě mají nedostatky, které jsou každým rokem hůře překonatelné. K vyřešení výše uvedených problémů je nutná nová technologie nebo nový přístup k budování informačních sítí. Taková technologie se nazývá – SDN (Software Defined Networking).

Hlavním cílem teoretické části diplomové práce je zhodnocení současného vývoje v oblasti SDN a možnosti programovatelnosti SDN prvků pomocí API (Application Programming Interface).

V první kapitole je obecně popsána SDN a hlavní důvody použití SDN oproti tradičním počítačovým sítím. Dále je vylíčena architektura SDN a zmíněny tři úrovně architektury. Poté je uváděna komunikace mezi úrovněmi architektury SDN a protokoly, které využívají. Také jsou zde popsány vlastnosti SDN kontrolérů, výhody a nevýhody využívaní SDN vytvořené na základě Open SDN, založené na existujících API i na základě superponovaných sítí a hypervisorů.

V druhé kapitole jsou vylíčeny typické příklady protokolů SDN a API. Poté je zde zmíněné rozhraní API, zejména REST (Representational State Transfer) API a nástroje pro nastavení. V nástrojích pro nastavení je probráno, jak lze na nejvyšší úrovni rozlišit hlavní rozdíly v architektuře nástrojů. Následně je pak popsán jeden z populárních nástrojů – Ansible.

V následující kapitole je nejen analyzován současný vývoj v oblasti SDN, ale obsahuje i popis nejlepších poskytovatelů a řešení SDN. Dále jsou pomocí analytických dat od společnosti Forrester a Gartner zjištěny čtyři nejlepší hardwarové platformy pro SDN v sítích datových center. Na konci kapitoly jsou popsány budoucí trendy SDN a směr vývoje.

Poslední kapitola je věnována navržení hlavního praktického cíle – laboratorních úloh. Úlohy se budou zabývat programováním SDN API. K laboratorním úlohám je vytvořen návod, který obsahuje základní teoretický úvod a postup práce. Cílem úloh je ukázat možností programovaní SDN API pomocí Pythonu a Ansible s využitím REST API v první úloze a eAPI v druhé.

V první laboratorní úloze bude využito hardwarové řešení BIG-IP od společnosti F5 Networks.

Ve druhé úloze budou využity virtuální směrovače od společnosti Arista Networks, které budou zprovozněny pomocí prostředí EVE-NG.

1 Softwarově definované sítě

V současné době jsou počítačové sítě strategickým faktorem ve vývoji moderních IT technologií. Bohužel síťová architektura není vždy schopna adekvátně a účinně reagovat na nové potřeby. Několik faktorů vede k tomu, že tradiční počítačové sítě mají nedostatky, které je každý rok těžké překonat. Mnoho tradičních sítí je hierarchických, ale taková statická architektura je špatně přizpůsobena potřebám dynamických výpočetních a úložných systémů. Mezi klíčové počítačové trendy, které určují potřeby nového síťového modelu, patří: změna schématu provozu, cloudové služby, mobilní zařízení, zvýšení objemu dat atd. Dají se rozlišit následující problémy moderních počítačových sítí:

- Vědecké a technické – dnes není možné řídit a spolehlivě předvídat chování komplexních objektů, jako jsou globální počítačové sítě.
- Ekonomické – sítě jsou drahé, složité a vyžadují pro své služby vysoce kvalifikované odborníky.
- Rozvojové problémy – v architektuře moderních sítí existují značné překážky při experimentování a vytváření nových služeb.

Odpovědí na krizi počítačových sítí byl vznik zásadně nového přístupu k jejich konstrukci – SDN. SDN je nová síťová architektura, ve které je správa sítě oddělena od přenosu dat a je přímo programovatelná. Tato migrace řízení, dříve úzce propojená v každém síťovém zařízení s přístupnými výpočetními zařízeními, umožňuje aplikacím a síťovým službám abstrahovat od základní infrastruktury a považovat síť za logickou nebo virtuální entitu. Hlavní důvody použití SDN oproti tradičním počítačovým sítím jsou následující:

- Oddělení procesu přenosu a správy dat.
- Jediné, na dodavateli nezávislé, rozhraní mezi řídicí vrstvou a vrstvou pro přenos dat.
- Logicky centralizovaná správa sítě, prováděná pomocí kontroléru s nainstalovaným síťovým operačním systémem a síťovými aplikacemi.
- Virtualizace fyzických síťových zdrojů [1].

Dále bude uvažována typická architektura SDN.

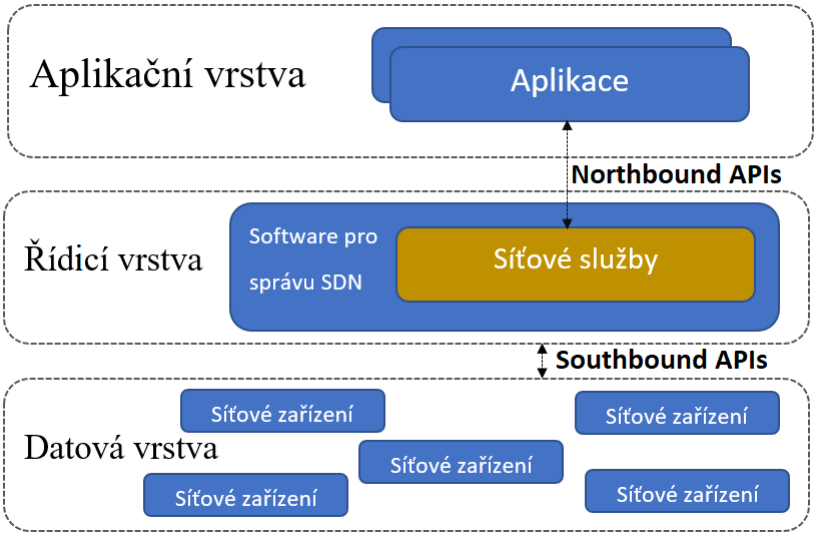
1.1 Architektura SDN

Architektura SDN má tři úrovně viz Obr. 1.1:

- Úroveň infrastruktury (datová vrstva), včetně sady síťových zařízení (přepínače a směrovače).
- Řídicí vrstva, která zahrnuje síťový operační systém, který poskytuje aplikacím síťové služby a softwarové rozhraní pro správu síťových zařízení a sítě.

- Aplikační vrstva pro flexibilní a efektivní správu sítě.

Datová rovina zahrnuje síťová zařízení, která jsou odpovědná za efektivní přenos dat. Na této úrovni je síťové vybavení, jako v tradiční síti. Na rozdíl od tradiční sítě jsou však zařízení fyzická i virtuální. Tato zařízení jsou pouze prostředkem pro odesílání datových proudů a nemohou se samostatně rozhodovat. Schopnost rozhodovat se přenáší na řídicí rovinu. SDN je navíc postavena koncepčně na otevřených a stan-

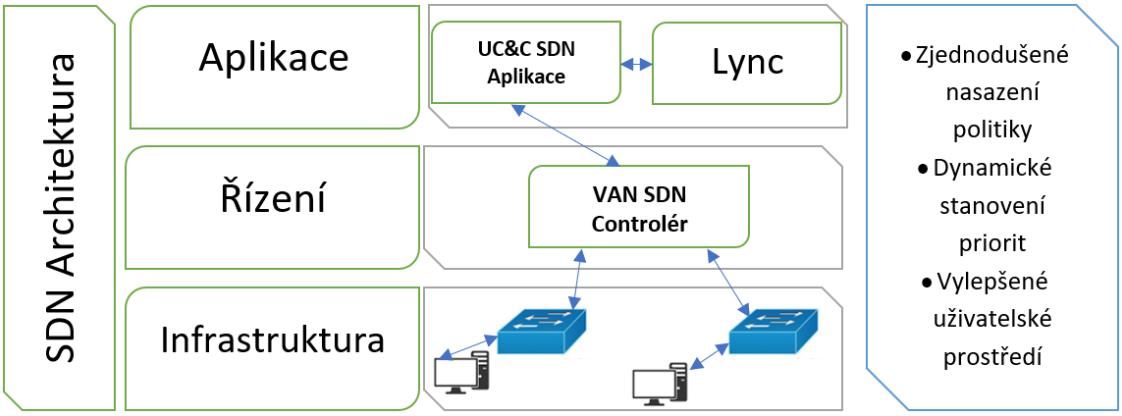


Obr. 1.1: Architektura SDN.

dardních rozhraních, například OpenFlow. Otevřená rozhraní umožňují kontrolérům dynamicky programovat heterogenní směrovače, což je v tradičních sítích obtížné kvůli široké škále uzavřených rozhraní a distribuované povaze řídicí roviny [2].

Řídicí rovina používá protokoly, skrze které jsou tabulky „flow“ naplněny v síťových prvcích datové roviny. Platforma pro správu zahrnuje softwarové služby používané k dálkovému sledování a nastavení funkcí správy sítí. Síťová politika je definována v řídicí rovině, řídicí rovina implementuje zásadu a datová rovina ji implementuje zasíláním dat v souladu s touto zásadou. V tradičních IP (Internet Protocol) sítích jsou řídicí a datové roviny úzce propojené i zabudované do stejných síťových zařízení a celá struktura je vysoce decentralizovaná. Výsledkem je velmi složitá a relativně statická architektura. To je také základní důvod, proč jsou tradiční sítě konzervativní a složité pro správu a kontrolu. Výhody použití SDN jsou dosaženy oddělením datové roviny od řídicí roviny. Díky oddělení řídicích rovin a dat se síťové přepínače stávají jednodušími předávacími moduly a řídicí logika je implementována v logicky centralizované kontroléry. Jak je vidět z architektury, kromě klasické správy sítě přímými příkazy správce systému ke kontroléru, SDN kontrolér podporuje také samotné spouštění aplikací pro správu sítě [2].

Každá aplikace v SDN je ve skutečnosti rozhraním pro optimalizaci sítě pro konkrétní obchodní aplikaci, například Microsoft Lynk Obr. 1.2, a její hlavní úlohou je změnit síť v reálném čase podle aktuálních potřeb obsluhovaného programu. V případě Microsoft Lynk to může být například změna v síti QoS (Quality of Service) mezi dvěma telefonními účastníky za účelem poskytování videohovoru HD (High-definition) v reálném čase bez zpoždění nebo vytvoření tunelu VPN (Virtual Private Network) mezi dvěma účastníky. Kontrolér SDN určuje datové toky, které



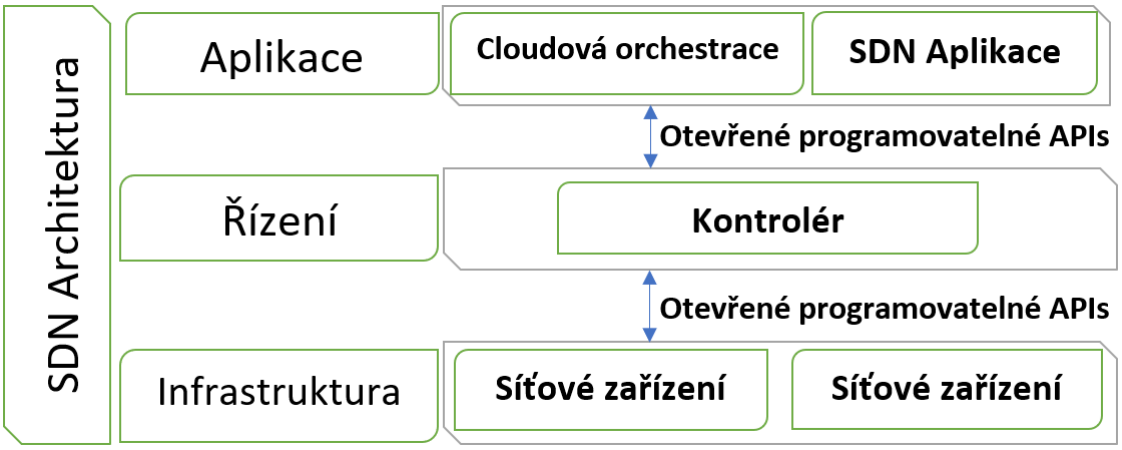
Obr. 1.2: SDN aplikace pro Microsoft Lync.

existují v datové rovině. Každý provoz v síti musí být nejprve povolen kontrolérem, který kontroluje, zda provoz neporušuje zásady sítě. Pokud kontrolér umožňuje provoz, vypočítá trasu provozu a přidá položku pro tento provoz do každého přepínače v cestě. Na rozdíl od složitých funkcí, které tvoří řídicí jednotku, přepínače jednoduše přeposílají rámce podle tabulek, které mohou být vyplněny pouze kontrolérem.

Podrobnější zabývání informačními proudy v architektuře SDN se dá rozdělit do dvou hlavních oblastí výměny informací. První je mezi aplikacemi SDN a druhá je pro správu fyzických síťových zařízení Obr. 1.3. První proud byl nazýván „Northbound“ rozhraní a druhý „Southbound“ rozhraní.

Interakce řídicí roviny a datové roviny se provádí pomocí „Southbound“ rozhraní. Nejznámějším protokolem pro interakci je OpenFlow, ale kromě toho existují i další: ForCES, OVSDB (Open vSwitch Database), POF OpFlex, OpenState, ROFL (Revised OpenFlow Library), HAL (Hardware Abstraction Layer), PAD (Programmable Abstraction of Datapath) [2].

Interakce řídicí roviny a aplikační roviny se provádí pomocí „Northbound“ rozhraní. Každý kontrolér má obvykle své „Northbound“ rozhraní, protože standardizace pro „Northbound“ rozhraní dosud nebyla vyvinuta. Například kontroléry jako Floodlight, Trema, NOX, Onix a OpenDaylight používají vlastní „Northbound“ APIs [2].



Obr. 1.3: Struktura a komponenty SDN.

Architektura SDN je extrémně flexibilní, je možné pracovat s různými typy přepínačů a na různých úrovních protokolu. Kontroléry a přepínače SDN lze implementovat pro ethernetové přepínače (vrstva 2), směrovače (vrstva 3), transport (vrstva 4) nebo směrování na úrovni aplikace. SDN se spoléhá na běžné funkce nalezené na síťových zařízeních, které se týkají hlavně předávání paketů na základě definice provozu. V architektuře SDN přepínače vykonávají následující funkce:

- Přepínač zapouzdřuje a přesměruje první paket provozu do kontroléru SDN, takže se správce může rozhodnout, zda přidat konkrétní provoz do tabulky „flow“ přepínače.
- Přepínač přeposílá příchozí pakety z odpovídajícího portu na základě „flow“ tabulky. Tabulka „flow“ může obsahovat informace o prioritě stanovené kontrolérem.
- Přepínač může zahodit pakety v konkrétním provoze dočasně nebo trvale, jak stanoví kontrolér. Vyhození paketů může být použito z bezpečnostních důvodů, aby se zabránilo útokům typu DoS (Denial of Service) nebo požadavkům na řízení provozu.

Tímto způsobem kontrolér SDN řídí stav přepínačů v SDN sítích. Tato správa se provádí pomocí API, které umožňuje kontroléru splnit nejrůznější aplikační požadavky, aniž by došlo ke změně jakýchkoli aspektů nižší úrovně sítě. Díky oddělení řídicích a datových rovin umožňuje SDN aplikacím pracovat s jedním abstraktním síťovým zařízením bez obav o detaily zařízení. Síťové aplikace vidí jenom API rozhraní kontroléru, tím pádem se dají rychle vytvářet a nasazovat nové aplikace a organizovat síťové provozy v souladu se specifickými požadavky na výkon a zabezpečení společnosti [3].

1.2 Vlastnosti SDN kontroléru

Pro porovnání kontrolérů mezi sebou lze rozlišit následující charakteristiky:

1. Cena a licence
   Náklady na kontroléry jsou rozděleny do dvou skupin: komerční a open source.
2. Účinnost
   Výkon řídicí jednotky je obecný pojem používaný k označení různých parametrů – výkonu, škálovatelnosti, spolehlivosti a zabezpečení. Různé indikátory, například počet rozhraní, které může kontrolér zpracovat, latence, šířka pásma atd. Tyto indikátory definují to, co se dá nazývat výkonem. Podobně existují různé metriky, které určují škálovatelnost, spolehlivost a zabezpečení. Většina práce na porovnání kontrolérů berou v úvahu pouze výkonnostní kritéria.
   Výkon závisí také na podporovaných programovacích jazycích. Python, C/C++ a Java jsou nejčastěji používanými jazyky pro programování kontrolérů SDN. Obecně platí, že kontroléry napsané v jazyce Java jsou multiplatformní a dobře modulární, ty, které jsou napsané v C/C++ poskytují vysoký výkon, ale nemají vysoký stupeň modularity, dobrou správu paměti a dobré grafické rozhraní. Kontroléry napsané v jazyce Python postrádají od skutečného více vláknového zpracování.
3. Centralizace a distribuce
   Centralizovaný kontrolér je jedna entita, která řídí všechna zařízení pro předávání v síti. Přirozeně to představuje jediný bod selhání a může mít omezené škálování. Při řízení sítě s velkým počtem prvků může nastat problém, že bude potřeba víc než jeden kontrolér.
   Na rozdíl od centralizovaného designu lze distribuované NOS (Network Operating System) přizpůsobit potřebám potenciálního prostředí, od malých až po velké sítě. Distribuovaný kontrolér může být centralizovaná skupina uzlů nebo fyzicky distribuovaná sada prvků. Ačkoli první z nich může nabídnout vysokou propustnost pro velmi velká datová centra, druhá z nich může být odolnější vůči různým typům logických a fyzických poruch. Poskytovatel cloudu, který pokrývá více datových center propojených pomocí globálních sítí, může vyžadovat hybridní přístup se shluky kontrolérů uvnitř každého datového centra a distribuovanými kontroléry na různých místech.
4. „Northbound“ rozhraní
   API pro „Northbound“ rozhraní používá aplikační vrstva pro interakci s kontrolérem. Jsou nejdůležitější součástí architektury kontrolérů SDN, protože účel SDN souvisí s inovativními aplikacemi. Kvůli tomu, že jsou dostatečně kritická, musí API orientovaná na „Northbound“ rozhraní podporovat široký rozsah aplikací. Tato API by také měla umožňovat připojení k automatickým zásob-

níkům, jako je OpenStack nebo CloudStack, používaných ke správě cloudu. V nedávné době se ONF (Open Networking Foundation) zaměřila na API pro „Northbound“ rozhraní po standardizaci „Southbound“ rozhraní (OpenFlow). Vytvořili pracovní skupinu „Northbound“, která bude psát kód, vyvíjet prototypy a tvořit standard pro „Northbound“ rozhraní. Protokol REST je v současnosti nejčastěji používaným „Northbound“ rozhraním a většina kontrolérů ho implementuje.

Vlastnost programovatelnosti sítě závisí také na podpoře „Northbound“ rozhraní. Nejdůležitější výhodou implementace SDN pro řešení složitých úkolů správy v moderní síti s velkým počtem připojených zařízení a zavádění nových služeb.

Aplikace lze nasadit přes platformu kontrolérů k provádění předdefinovaných úkolů a funkcí správy. Schopnosti kontroléru v síťovém programování jsou určovány hlavně stupněm integrace velkého počtu „Northbound“ rozhraní, dobrým grafickým uživatelským rozhraním a rozhraním příkazového řádku.

5. „Southbound“ rozhraní a podpora OpenFlow
   Pro interakci kontroléru s předávacími zařízeními se používají protokoly „Southbound“ rozhraní (OpenFlow, OVSDB, ForCES) a rozšíření existujících protokolů pro správu fyzických nebo virtuálních zařízení, například SNMP (Simple Network Management Protocol), BGP (Border Gateway Protocol), NetConf (Network Configuration Protocol). Použití „Southbound“ rozhraní je důležité jak pro zpětnou kompatibilitu, tak pro podporu heterogenních sítí. V současné době je však jedním z nejpopulárnějších protokolů „Southbound“ rozhraní OpenFlow [4].

1.3 Výhody a nevýhody využívaní SDN

Softwarově konfigurovatelné síťové technologie se celosvětově aktivně vyvíjejí. Pojem SDN je obecný koncept, konkrétní implementace SDN se mohou výrazně lišit v jejich architektuře. Rozlišujeme tři oblasti implementací SDN:

- SDN vytvořený na základě původní, originální verze navržené odborníky na Stanfordské univerzitě v roce 2007.
- SDN vytvořené na základě existujících API.
- SDN vytvořené na základě superponovaných sítí a hypervizorů.

První směr, je implementace SDN v původní verzi. Zahrnuje přesun funkce správy sítě ze síťových zařízení na centralizovaný kontrolér pomocí protokolu OpenFlow. Takový SDN by měl mít, jak věřili jeho iniciátoři, následujících pět základních vlastností: oddělené úrovně správy, používat jednodušší síťová zařízení, provádět centralizovanou správu sítě, používat automatizaci a virtualizaci síťových funkcí, být

otevřeno odborníkům a vývojářům. Pro takové sítě s uvedenými vlastnostmi byl vytvořen zvláštní termín „Open SDN“. Povinným prvkem takové SDN je přítomnost kontroléru pracujícího se sítí prostřednictvím rozhraní OpenFlow.

Ve druhém směru se při vytváření SDN používají funkce API, které lze vyvolávat vzdáleně, obvykle pomocí tradičních mechanismů, jako je SNMP, CLI (Command Line Interface), nebo při použití novějších, flexibilnějších mechanismů, jako je REST API.

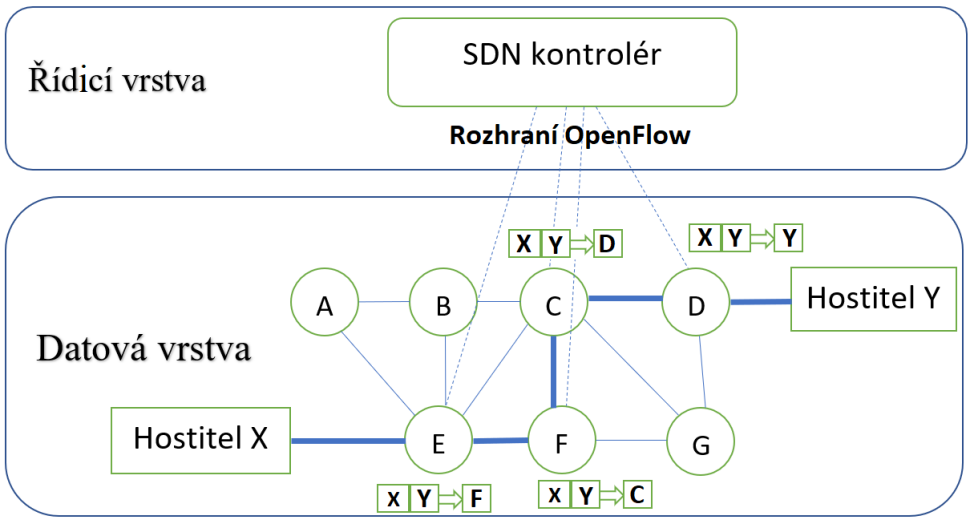
Implementace SDN ve třetím případě nezávisí na základní síťové infrastruktuře. SDN je položena na horní část existující fyzické sítě. Druhá i třetí možnost vytváření SDN jsou alternativy k Open SDN, ale v některých případech jsou výhodnější při vytváření počítačových sítí. Při vytváření počítačové sítě je potřeba zvážit správnost výběru jedné nebo druhé možnosti.

1.3.1 SDN vytvořené na základě Open SDN

Na začátku se objevil koncept Open SDN v původní, originální interpretaci s velkým nadšením. Sítě Open SDN mají skutečně mnoho výhod, díky odstranění úkolů správy dat z přepínačů je pohyb provozu zrychlen, což výrazně zvyšuje produktivitu. Současně se díky virtualizaci správy sítí sníží náklady na jejich výstavbu a údržbu. Na centralizovaném otevřeném kontroléru Open SDN může správce systému sledovat celou síť v jediném pohledu, což zlepšuje pohodlí správy, zabezpečení a zjednodušuje řadu dalších úkolů. Teoreticky neomezené možnosti rozšíření sítí Open SDN umožňují vytvářet skutečné cloudy, které jsou škálovatelné v závislosti na úkolech. Síť má zároveň potřebnou inteligenci, která je nezbytná zejména pro organizaci práce velkých skupin přepínačů.

Brzy však byla spousta kritiky spojena s architekturou Open SDN a s otázkami praktického zavedení takových sítí. Nejvýznamnějšími kritiky jsou výrobci síťových zařízení, kteří se na změny, které nová síťová technologie přináší, dívají s velkým despektem. Jedná se o značné náklady na nové vybavení, jako jsou přepínače, které podporují rozhraní OpenFlow, a rizika vyplývající z nedostatečného testování nového zařízení, které by zákazníci měli masivně implementovat. Zvýší se také náklady na rekvalifikaci mnoha IT specialistů, kteří budou pracovat s novými sítěmi. Nejzávažnějším argumentem kritiků Open SDN, z hlediska její architektury, je zranitelnost takové sítě kvůli přítomnosti jediného bodu selhání. Obrázek 1.4 ukazuje diagram provozu dat v Open SDN, který je spravován jediným kontrolérem. Pokud tento kontrolér selže, celá síť je nefunkční. To znamená, že ilustrovaný SDN kontrolér je jediný bod selhání [5].

K překonání této nevýhody je nutné mít v síti několik kontrolérů, které vzájemně komunikují pomocí zvláště spolehlivých komunikačních linek. Tento faktor



Obr. 1.4: Propagace dat v síti v souladu s architekturou Open SDN.

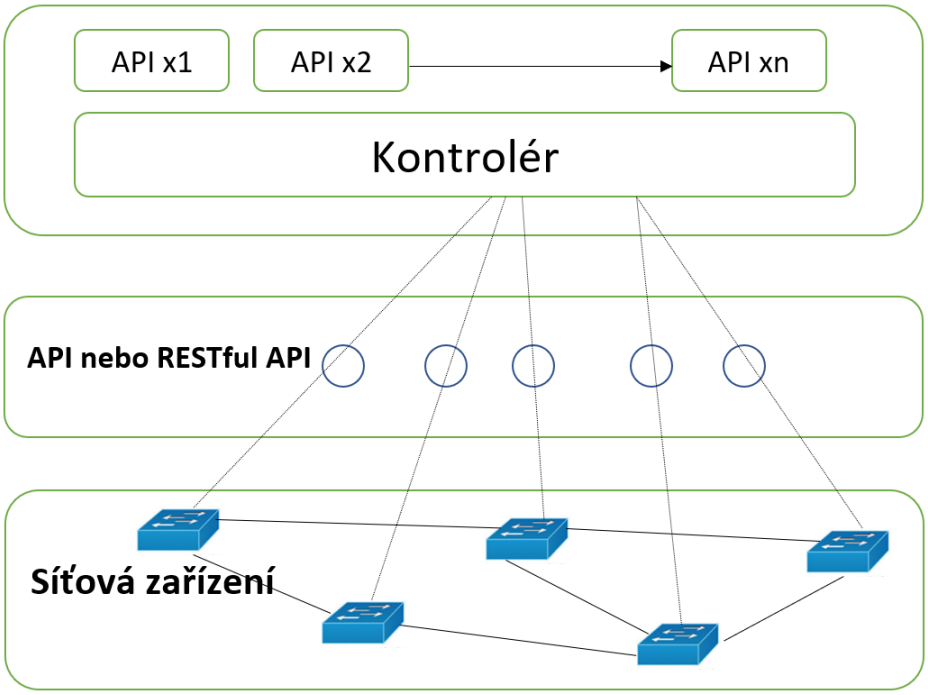
komplikuje a zvyšuje náklady na architekturu Open SDN.

1.3.2 SDN založené na existujících API

Pokud síťovým zařízením výrobci umožní rozpoznání širší sady příkazů API, pomocí kterých může regulátor flexibilně řídit zařízení a celou síť, bude to SDN implementovaný prostřednictvím existujících API Obr. 1.5.

Někteří výrobci upgradují existující API na zařízeních, například místo tradičních CLI a SNMP se implementuje REST API. Mechanismy CLI a SNMP byly dlouho vyvíjeny a používány v síťových nastaveních, ale v současné době, kdy je nutné rychle a dynamicky spravovat velkou síť nebo datové centrum, jsou tyto mechanismy příliš objemné a nepohodlné. Proto byly nahrazeny novým mechanismem – REST API. V posledních letech se tento mechanismus stal nejčastějším při přenosu požadavků pro API rozhraní přes síť. Technologie REST API funguje pomocí protokolu HTTP (Hypertext Transfer Protocol) pro přenos hypertextu, který se obvykle používá na webu. Technologie REST API je relativně jednoduchá a snadno rozšiřitelná. Je vhodné ji používat, protože používá standardní port TCP (Transmission Control Protocol), který nevyžaduje speciální nastavení brány firewall, aby požadavky REST API mohly procházet. SDN vytvořené z existujících API mají několik výhod. Jednou ze zřejmých výhod je to, že pracují s běžnými, neaktualizovanými přepínači. To znamená, že není nutné zavádět nový typ přepínače s podporou standardu OpenFlow.

Další výhodou tohoto přístupu je, že flexibilita správy sítě je do určité míry vylepšena. Stávající API zjednodušují poznámkový software pro organizování událostí



Obr. 1.5: SDN na základě existujících API.

v síti. To znamená rychle a automaticky reagovat na změny v síti, například dynamický pohyb virtuálních zařízení v datových centrech. Další výhodou je, že použití dostupných API umožňuje vybudovat síť s centralizovaným řízením v rámci určitých limitů. To vede k větší otevřenosti v síťových architekturách, protože výrobci jsou nuceni otevírat specifikace rozhraní svého proprietárního vybavení. Tento proces je nezbytný pro vývoj a normální provoz aplikací vývojáře třetích stran.

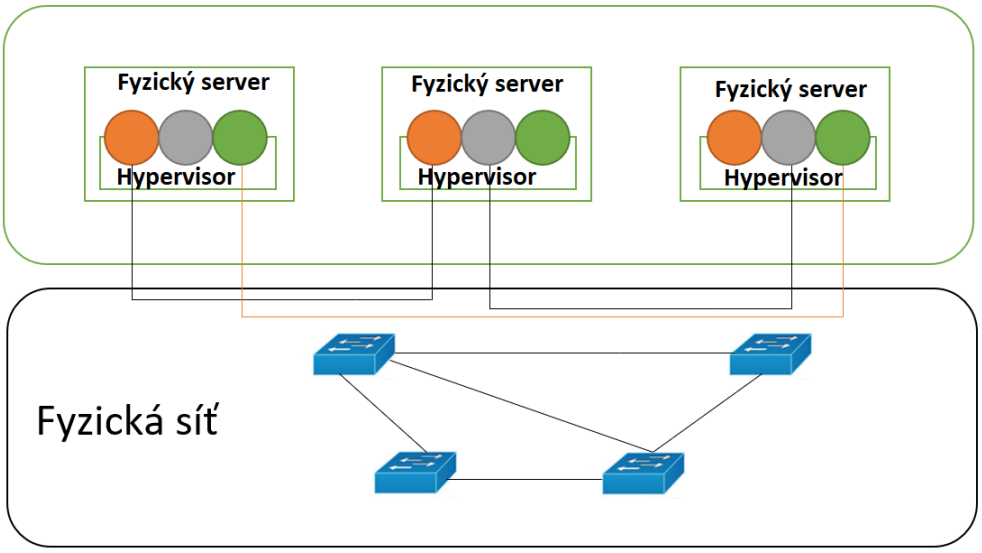
SDN vytvořené z existujících API mají ale i své nevýhody. Jednou z nich je, že v takových sítích je ve většině případů kontrolér zcela nepřítomen. Proto je síťový programátor nucen programovat každý přepínač přímo, ale i když existuje kontrolér, programátor nemá společný standardní mechanismus pro interakci se síťovými zařízeními. To znamená, že programátor je nucen znát technické vlastnosti rozhraní každého přepínače. Další nevýhodou je, že software takové SDN bude fungovat pouze pro konkrétní konfiguraci sítě, což je pochopitelné, protože API zařízení různých výrobců nespadají pod obecný standard, na rozdíl od protokolu OpenFlow. Proto je tento typ SDN schopen pracovat se zařízením konkrétního výrobce nebo s malou skupinou výrobců kompatibilních zařízení. Další nevýhodou je, že pohyb řídicích funkcí z přepínače do kontroléru předepsaného v architektuře SDN byl zaměřen na vytvoření jednoduchých, levnějších přepínačů. Tento problém nelze vyřešit, protože SDN byla vytvářena na základě stejných složitých a drahých přepínačů, i když jsou SDN vytvořené na základě existujících API. Taková API neumožňují kontrolovat přenos

dat do určité míry, zejména při vytváření VLAN a VPN, nemohou zajistit lepší kontrolu každého datového toku, jak je to možné pomocí protokolu OpenFlow [5].

SDN vytvořené na základě existujících API lze považovat za praktické rozšíření současné funkčnosti počítačové sítě, pokud radikálnější řešení založené na OpenFlow není z nějakého důvodu k dispozici nebo nepraktické. Celkově lze říci, že vytvoření SDN založených na existujících API je krokem správným směrem, krokem směrem k vytvoření sítě s plně centralizovanou správou programů.

1.3.3 SDN na základě superponovaných sítí a hypervisorů

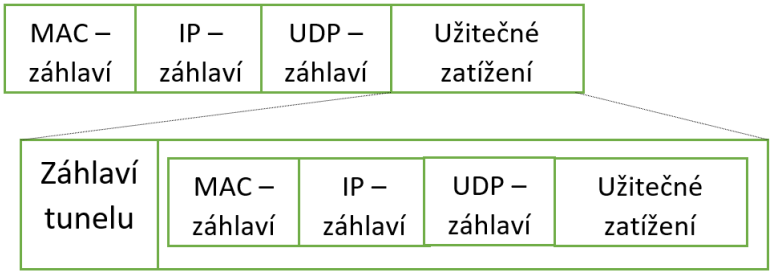
Alternativním a inovativnějším způsobem vytváření SDN je použití superponovaných sítí řízených hypervizory. V takovém případě zůstává fyzická síť beze změny. Nad touto fyzickou sítí jsou však vytvořeny superponované virtuální sítě s hypervizory. Aplikační systémy v síťových uzlech interagují s těmito virtuálními sítěmi a nevědí nic o fyzických vlastnostech sítě, přes kterou jsou data přenášena Obr. 1.6. Protože virtuální sítě umístěny nad fyzickou infrastrukturou, mohou být řízeny sys-



Obr. 1.6: Virtuální sítě umístěné na fyzické síti.

témy (nebo zařízeními) umístěnými na koncových uzlech sítě. V datových centrech jsou takové systémy hypervizory virtuálních zařízení, které jsou přítomny na každém serveru. Přenos provozu ve virtuálních sítích se provádí tunelováním pomocí zapouzdření. To znamená, že když paket dorazí do virtuálního síťového uzlu pro přenos, síťové zařízení (obvykle hypervisor) zapouzdří tento paket v jiném rámci Obr. 1.7 [5].

SDN postavené na superponovaných sítích a hypervizorech fungují dobře v datových centrech, kde je již nainstalován software pro virtualizaci serverů. Pomáhají eliminovat řadu problémů, které se vyskytnou během provozu datového centra. Nej-



Obr. 1.7: Zapouzdření datového paketu v rámci tunelu.

prve je eliminován explozivní růst MAC (Media Access Control) adres uzlů, protože MAC adresy jsou při použití SDN skryté v zapouzdřených rámcích. Na omezení počtu podporovaných sítí VLAN v lokálních sítích již nezáleží, protože zde se tunelování (namísto VLAN) používá k oddělení provozu více datových toků. Konstrukce datových center založená na SDN umožňuje velmi flexibilně a rychle změnit vlastnosti sítí zapojených do výpočetních procesů datového centra, díky možnosti centralizované správy softwaru virtuálních sítí SDN [5].

SDN vytvořené na základě superponovaných sítí a hypervizorů nevyřeší všechny problémy. Zejména fyzická síťová infrastruktura stále vyžaduje ruční konfiguraci a údržbu, což platí například pro QoS, STP (Spanning Tree Protocol). Další nevýhodou je, že v takových SDN, stejně jako v předchozí verzi, zůstávají síťová zařízení beze změny, neaktualizují se a nezjednodušuje se jejich správa.

2 Programovatelnost SDN

V kapitole 1.1 byla zmíněna problematika „Southbound“ a „Northbound“ rozhraní. Dále byly také zmíněny protokoly, které se v SDN sítích používají. Tabulka 2.1 se pokouší zjednodušit porozumění různým API, protokolům SDN a jejich zamýšlenému použití.

Tab. 2.1: Typické příklady protokolů SDN a API

	Typ					
Název	Protokol	API	Rámec	Rozhraní	Účel	Typické SDN kontroléry
ForCES	+		+	Southbound	Komunikace	Specifické pro dodavatele
OpenFlow	+			Southbound	Komunikace	RYU, ONOS, Openvirtex, Opendaylight
Netconf		+		Southbound	Management	Opendaylight, ONOS, Opencontrail, RYU
OVSDB	+			Southbound	Konfigurace	Opendaylight, Opencontrail, RYU, ONOS
XMPP		+		Southbound	Protokol zpráv	Opencontrail
REST		+		Northbound	Konfigurace	Opendaylight, ONOS
BGP LS	+			Southbound	Komunikace	Opendaylight
BGP	+			Southbound	Komunikace	Opencontrail

Lze si všimnout, že ačkoliv jsou Opencontrail a OpenVirtex jsou uvedeny jako kontroléry SDN, jedná se spíše o síťový hypervisor nebo síťový virtuální kontrolér. Dále bude rozebráno více do hloubky pouze REST API a nástroje pro nastavení SDN, zejména Ansible.

2.1 API

API – je kombinace různých nástrojů a funkcí implementovaných jako rozhraní pro vytváření nových aplikací, pomocí kterých bude jeden program interagovat s druhým.

V případě vývoje mobilních aplikací může být API knihovnou. Všechny nuance budou implementovány v knihovně a k tomuto API se lze připojit pouze ve svém kódu. V případě webových aplikací může API vykreslit data v jiném formátu, než je standardní HTML, což usnadňuje použití při psaní vlastních aplikací [6].

2.1.1 Typy API

Existují tři druhy API:

- RPC (Remote Procedure Call) – vzdálené volání procedur.
- SOAP (Simple Object Access Protocol) – protokol jednoduchého přístupu k objektům.
- REST – je cesta, jak jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých HTTP volání.

Rozhraní API lze rozdělit podle typu služby, kde se využívají:

- aplikace,
- webové stránky,
- operační systémy.

Například většina operačních systémů (Unix, Windows, MacOS atd.) má API, které umožňuje programovat tento systém.

Rozhraní API lze také rozdělit podle typu přístupu:

- Interní APIs (dostupná interním vývojářům a zaměstnancům společnosti, která se používají k optimalizaci pracovních procesů a snižování nákladů).
- Partnerská APIs (dostupná obchodním partnerům a spotřebitelům produktu nebo služby, která se používá k optimalizaci procesů a vývoje).
- Veřejná APIs (dostupná pro všechny, používaná k vytváření nových služeb a popularizaci stávajícího směru) [6].

2.1.2 Výhody a nevýhody práce s API

Výhody používání API:

- Hlavní výhodou práce s API je úspora času při vývoji vlastních služeb. Programátor dostává hotová řešení a nemusí trávit čas psaním kódu pro funkčnost, která byla implementována po dlouhou dobu.
- API může zohlednit nuance, které vývojář třetí strany nemusí vzít v úvahu, nebo je prostě nemusí vědět.
- API poskytuje aplikacím určitou systematičnost a předvídatelnost. Proto ho lze implementovat do různých aplikací způsobem, který je srozumitelný a známý všem uživatelům.
- API poskytuje vývojářům třetích stran přístup k proprietárním službám.

Nevýhody používání API:

- Pokud jsou provedeny změny a vylepšení hlavní služby, nemusí být v API okamžitě k dispozici.
- Vývojář má k dispozici hotová řešení, ale jak jsou implementována a jak vypadá zdrojový kód, neví.
- API je primárně určeno pro všeobecné použití, nemusí být vhodné pro vytvoření určité speciální funkce [6].

2.2 REST

REST je styl softwarové architektury pro distribuované systémy, jako je World Wide Web, který se obvykle používá k vytváření webových služeb. Termín REST zavedl v roce 2000 Roy Fielding, jeden z autorů protokolu HTTP. Systémy podporující REST se nazývají RESTful systémy.

Obecně je REST velmi jednoduché rozhraní pro správu informací bez použití jakýchkoli dalších vnitřních vrstev. Každá informace je jedinečně identifikována globálním identifikátorem URI (Uniform Resource Identifier). To znamená, že adresa URI je v podstatě primárním klíčem pro jednotku dat.

Jak jsou spravovány servisní informace, je zcela založeno na protokolu přenosu dat. Nejběžnějším protokolem je samozřejmě HTTP. Pro HTTP je akce na datech nastavena pomocí metod: GET (získat), PUT (přidat, nahradit), POST (přidat, změnit, odstranit), DELETE (odstranit). Akce CRUD (Create-Read-Updtae-Delete) lze tedy provádět všemi čtyřmi metodami a to pouze pomocí GET a POST [7].

2.2.1 RESTful

Většina API, která dnes existují v síťové infrastruktuře jsou HTTP RESTful API. Aby byl distribuovaný systém považován za RESTful, je třeba se ujistit, že splňuje následující kritéria:

- Client-Server – systém by měl být rozdělen na klienta a server.
- Stateless – server by neměl ukládat žádné informace o klientovi.
- Cache – každá odpověď musí být označena, zda je možné ji uložit do mezipaměti nebo ne, aby se zabránilo klientům v opakovaném použití zastaralých nebo nesprávných údajů v reakci na další žádosti.
- Uniform Interface – jedno rozhraní definuje rozhraní mezi klientem a serverem.
- Layered System – v REST je povoleno rozdělit systém do hierarchie vrstev, avšak s podmínkou, že každý komponent uvidí pouze komponenty další vrstvy.

- Code-On-Demand(volitelné) – REST umožňuje stahovat a spouštět kód nebo programy na straně klienta [8].

Pokud se jedná o síťové zařízení nebo kontrolér SDN, znamená to, že takové RESTful API poskytuje výměnu informací mezi klientem a serverem. Klient je obvykle reprezentován aplikací, jako je skript Pythonu nebo webová aplikace s uživatelským rozhraním, a server je síťové zařízení nebo kontrolér. Protože protokol HTTP se používá jako přenosový protokol, musí uživatel provádět některé operace s adresou URL (Uniform Resource Locator), stejně jako při surfování po internetu. Při přístupu na web a při vyplňování jakéhokoliv webového formuláře se provádí operace HTTP GET, po kliknutí na tlačítko odeslat je provedena operace HTTP POST. Z těchto znalostí by mělo stačit rozumět práci s RESRTful API. Víc o tom, jak probíhá komunikace mezi webovým prohlížečem a web serverem, je zobrazeno na Obr.2.1. Z obrázku se dá také pochopit to, že komunikace mezi API klienta a serverem přes Postman probíhá velice podobně první komunikaci. Hlavní rozdíl, což

Obr. 2.1: Reprezentace konceptů RESTful API.

je taktéž zobrazeno na Obr.2.1, je v tom, že přenášená data v prvním a druhém případu jsou různá. Od webového prohlížeče se dají obdržet HTML data, která jsou interpretována prohlížečem tak, aby správně zobrazovala vzhled a obsah webu. Pokud je požadavek HTTP GET adresován na webový server, který podporuje rozhraní RESTful API, jsou pak vrácená data obvykle kódována ve formátu JSON (JavaScript Object Notation) nebo XML (Extensible Markup Language). JSON je mnohem stručnější a snadněji čitelný než XML formát [9].

2.3 Nástroje pro nastavení

Nejvíce jsou využívány následující nástroje:

- Ansible,
- Salt,
- StackStorm.

Každý nástroj má své výhody a nevýhody. Tato část stručně popisuje všechny nástroje, a jak mohou být použity v konkrétním síťovém prostředí. Na nejvyšší úrovni lze rozlišit následující hlavní rozdíly v architektuře:

- Použití agentů nebo nedostatek agentů – některé nástroje vyžadují agenty, malé kousky softwaru, které běží na spravovaném systému nebo zařízení. V souvislosti s automatizací sítě to může být problém, protože ne každý síťový operační systém podporuje fungování agentů na síťovém zařízení. V případech, kdy síťový operační systém nepodporuje činnost agenta na zařízení pomocí vlastních prostředků, se někdy k vyřešení tohoto problému ve formě proxy agenta používají další metody. Nástroje z opačné skupiny nepotřebují agenta, takže mohou být vhodnějšími nástroji pro automatizaci sítě.
- Centralizace nebo decentralizace – u architektur založených na používání agentů je často také vyžadován centralizovaný „hlavní server“. Některé neamerické produkty také implementují koncept „core server“, ale většina nástrojů bez agentů má decentralizovanou strukturu.
- Specializovaný protokol nebo standardní protokol – některé nástroje používají svůj vlastní specializovaný protokol, nejčastěji spojený s architekturami založenými na agentech. Jiné nástroje používají jako transportní protokol SSH (Secure Shell).
- DSL (Domain-specific Language) nebo standardní formáty dat a jazyky pro obecné použití – některé nástroje používají svůj vlastní doménový jazyk. V takovém případě by uživatelé měli vytvořit odpovídající soubory zpracované v nástroji v tomto jazyce. DSL je jazyk vytvořený pro řešení specifických problémů (dosažení určitých cílů) v určité oblasti (nebo pro konkrétní nástroj). Jiné nástroje používají YAML (Ain't Markup Language), který je v této souvislosti považován za univerzální standardní jazyk.
- Rozšiřující jazyk – většina z výše uvedených automatizačních nástrojů podporuje možnost přidávat nebo rozšiřovat funkce pomocí skriptovacího jazyka na vysoké úrovni. V některých nástrojích je jako jazyk pro rozšíření funkčnosti, vybrán Ruby, ostatní používají Python.
- „Push“ model, „pull“ model nebo model založený na událostech – některé nástroje vytvářejí svůj pracovní postup na základě modelu „push“, to znamená, že informace jsou přenášeny („push“ – „pull“) z jednoho centra na všechna

spravovaná zařízení nebo systém. Jiné nástroje používají model „pull“, podle kterého se informace o konfiguraci nebo instrukce obvykle přenášejí na vyžádání („pull“), nejčastěji na základě nějakého druhu plánovaného harmonogramu. Kromě toho existují nástroje řízené událostmi, které provádějí určité akce v reakci na výskyt určité události nebo při spuštění spouště [10].

2.3.1 Ansible

Ansible – má decentralizovanou architekturu bez použití agentů a používá SSH jako základní přenosový protokol. Obvykle funguje na základě modelu „push“, ale také podporuje model „pull“. Nástroj Ansible je napsán v jazyce Python a používá tento jazyk k rozšíření funkčnosti. Obsahuje podporu práce se šablonami napsanými v jazyce Jinja. Ansible byl původně využíván jako prostředek k rychlému provádění specializovaných příkazů na serverech, ale postupem času se z něj vyvinul výkonný nástroj pro orchestraci úkolů pomocí takzvaných „knih“, které provádějí typické úkoly s konstantními výsledky na cílových systémech. Knihy skriptů mohou být psány ve standardním jazyce YAML nebo v dialektu jazyka YAML specializovaného na Ansible [11].

Pokud se jedná o Ansible spolu s Cisco nebo Arista a hromady dalších hardwarů a OS (Operating System). To znamená, že Ansible už má v sobě syntax modulů, které lze využít. Seznam modulu se dá vždy najít na hlavním webu Ansible. Pokud nutný modul neexistuje, lze použít „knihu“ napsanou pomocí Jinja [12]. Tato „kniha“ většinou bude obsahovat CLI strukturu zařízení, pro které je napsána. Základní Ansible nemá GUI (Graphical User Interface), pouze CLI prostředí OS, na kterém je zprovozněn. Existuje Ansible Tower nebo AWX, což jsou produkty, které mají GUI, bohužel jenom AWX má licence zadarmo. Také je do dnes Ansible Tower ve fázi aktivního vývoje a vylepšení.

3 Současný vývoj v oblasti SDN

SDN zjednodušuje správu síťových zdrojů a funkcí a poskytuje programovatelné rozhraní, snižuje provozní náklady a také poskytuje automatizaci a flexibilitu sítě.

Z tohoto důvodu poskytovatelé služeb přecházejí na softwarově definované sítě. Podle poslední zprávy o analýze trhu, kterou připravila Data Bridge Market Research se očekává, že globální trh SDN do roku 2026 vzroste na odhadovanou hodnotu 67,98 miliard USD, přičemž v prognózovaném období 2019–2026 bude CAGR (Compound Annual Growth Rate) 28,90% [13].

Nejúspěšnějším bodem pro SDN byl nákup společnosti VMware společnosti Nicira v roce 2013. Poté VMware převzal obrovskou část trhu síťových technologií, nyní však NSX SDN společnosti VMware není založena na slibné technologii. Primární nabídkou SDN je infrastruktura zaměřená na Cisco, protože většina firem používají hardwary od Cisco. Společnosti jako Juniper, Nokia a další mají své produkty SDN. V této fázi vývoje IDC (International Data Corporation) odhaduje, že se SDN dostává do popředí. Network World v roce 2018 provedl průzkum mezi 300 lidmi, kteří mají rozsáhlé zkušenosti v oblasti sítí, a zjistili, že 19% SDN již používá a 50% SDN testuje. IDC činí 3 hlavní směry SDN v této fázi vývoje:

- Investice do virtualizace serverů a privátního cloudu.
- Zajištění programovatelnosti sítě.
- Zabezpečení.

3.1 Současní poskytovatelé řešení SDN

- Cisco ACI (Application Centric Infrastructure)
  V roce 2013 přední síťová společnost v oboru plně získala spuštění Insieme (v němž již Cisco mělo 85% akcií) a zahájilo novou architekturu datových center ACI, která je v tomto odvětví lídrem v oboru. Nyní se síť stala jedním z dominantních lídrů na trhu SDN. Cisco ACI nebo Application Centric Infrastructure je řešení SDN, které pracuje s kontrolérem infrastruktury aplikačních zásad Cisco APIC (Application Policy Infrastructure Controller) a Nexus 9000.
  - Výhody: jednoduchá obsluha, snadné programování, škálovatelnost.
  - Nevýhody: vysoké náklady na licence, nedostatek dokumentace, složitá implementace, obtížnost opravit potenciální problémy, integrace se sítěmi třetích stran nebo jinými protokoly je příliš komplikovaná nebo omezená.
- VMware NSX
  VMware NSX for Data Center je špičkovým řešením SDN v oboru. Společnost VMware koupila společnost Nicira v roce 2012 a o rok později uvedla

společnost VMware na trh první verzi NSX vydanou v říjnu 2013, přičemž se jedná o čisté síťové řešení, které může pracovat s jakýmkoli přepínačem. NSX poskytuje celou řadu síťových schopností ve virtualizované podobě. To znamená, že je možné vytvořit obrovskou superponovanou síť virtuálních směrovačů, hraničních bran, logických přepínačů, firewallů, vyrovnávačů zatížení, VPN s mnoha síťovými službami, aniž by bylo nutné používat jakoukoli proprietární hardwarovou součást sítě. VMware nabízí NSX ve dvou verzích, NSX-v a NSX-T, kde NSX-v je pouze pro hypervizora vSphere a NSX-T pracuje s nejmodernější infrastrukturou, jako je OpenStack, KVM, Kubernetes, OpenShift, AWS, Azure.
  - Výhody: Snadná integrace s více platformami s NSX-T, nejlepší GUI.
  - Nevýhody: žádné podkladové řešení.
- Nokia Nuage Networks

  Nokia, světový lídr v oblasti telekomunikací, nabízí síťové softwarové řešení Nuage Networks VSP (Virtualized Services Platform), které poskytuje nejmodernější síťovou automatizaci. VSP je založen na vlastním VSC (Virtualized Services Controller) a dvou dalších klíčových prvcích: VSD (Virtualized Services Directory) a VRS (Verification Router Service). Poskytuje celou řadu produktů pro optimalizaci výkonu datových center, cloud computingu a SDN.
  - Výhody: nejjednodušší produkt pro integraci do staršího vybavení.
  - Nevýhody: méně dokumentace, žádná otevřená komunita, složitá počáteční konfigurace.
- Juniper's Contrail Network

  Společnost Juniper Networks vstoupila na trh softwarově definované sítě se svým produktem Contrail Networking v roce 2013 po akvizici společnosti Contrail Systems v roce 2012. Souběžně byla komunita OpenContrail otevřeným zdrojem pro příjem vstupů od vývojářů. Avšak v roce 2018 bylo vlastnictví projektu OpenContrail převedeno na linuxovou nadaci a přejmenováno na Tungsten Fabric.

  Juniper Contrail Networking poskytuje připojení k jakémukoli typu aplikací běžících na virtuálních strojích, kontejnerech nebo dokonce otevřených zdrojových serverech přes superponovanou síť.
  - Výhody: jednoduché, otevřené standardy, vysoký rozsah a produktivita.
  - Nevýhody: nejsou díky tomu, že existuje otevřená komunita a je dost dokumentace.
- Big Switch Networks

  Big Switch Networks je americká cloudová a datová společnost založená v roce 2010. Jeho vlajkový produkt, BCF (Big Cloud Fabric), spolu se softwarem Switch Light, poskytuje síťový software založený na Openflow pro soukromé,

veřejné, hybridní cloud a více sítí.
  - Výhody: redukuje řídicí konzoli o více než 60 : 1, fyzická + virtuální struktura založená na otevřených principech SDN na každé úrovni implementace.
  - Nevýhody: nedostatek dokumentace, jediný bod selhání jsou kontroléry.
- Arista Networks
  Arista Networks se stala jednou z nejrychleji rostoucích společností v oblasti cloudových sítí. Svou práci zahájila začátkem roku 2004 a do dnešního dne nasadila na celém světě více než 20 milionů cloudových portů. Jádrem řešení Arista SDN je EOS (Extensible Operating System) založený na nezměněném linuxovém jádře. Směrovač s podporou Arista EOS a další virtuální zařízení umožňují uživateli spouštět více síťových aplikací, přepínání Ethernet a směrovací protokoly v překryvné síti s automatizovanými procesy IT.
  - Výhody: vynikající podpora, jakékoli cloudové API, vysoce stabilní a bohatý na funkce.
  - Nevýhody: verze pro seznámení je dost omezená.
- Cumulus Networks Cumulus Networks je softwarová společnost pro počítačové sítě, nejznámější pro svůj síťový operační systém založený na systému Debian Linux, Cumulus Linux. Tento síťový operační systém podporuje více než 30 platforem s otevřeným zdrojovým kódem od předních výrobců hardwaru. Společnost byla podporována několika zakladateli VMware a veřejně se objevila v červnu 2013. Společnost Cumulus Networks nedávno spolupracovala se společností Nutanix na poskytování hyperkonvergované infrastruktury s protokolem Cumulus SDN a také podepsala smlouvu se společností Facebook pro Minipack a stala se prvním síťovým operačním systémem, který podporuje Facebook od společnosti Minipack a účastní se projektu Open Compute Project.
- Pluribus Networks
  Pluribus Networks byl založen v roce 2010 třemi bývalými spolužáky ze Stanfordu a získal finanční prostředky od Yahoo! zakladatel Jerry Yang. Tito poskytovatelé SDN nabízejí řešení s odlišným přístupem. Společnost vyvinula novou generaci řešení SDN bez kontroléru, Adaptive Cloud Fabric a Netvisor ONE Open Network Operating System.
- Huawei
  Huawei již řadu let inovuje v SDN hardwaru a softwaru. Získala ocenění „Best SDN Solution“ v roce 2017. Vítězové soutěže byli stanoveni na základě pěti klíčových kritérií: reputace prodávajícího na trhu, vlastnosti a jedinečné odbytiště řešení, rozšířené uznávání technologií nebo značek, zpětná vazba uživatelů o řešení a nasazení, také trvanlivost, škálovatelnost a kvalita služeb. Ocenění

„Best SDN Solution“ je možné získat za řešení, která umožňují správcům sítě mít programovatelné centralizované řízení síťového provozu a prostředků, aniž by vyžadovali fyzický přístup k síťovým hardwarovým zařízením. Huawei SDN zahrnuje novou generaci kontroléru SDN „Agile Controller 3.0“, což je projekt otevřeného systému založeného na operačním systému ONOS (Open Network Operating System) a kontroléru Open Daylight. Řešení také zobrazuje snadno použitelná rozhraní a poskytuje významné výhody, jako je zkrácení doby uvádění nových služeb na trh.

3.2 Hardwarové platformy pro SDN v sítích datových center

Tato kapitola má za úkol určit čtyři nejlepší hardwarové platformy pro SDN v sítích datových center, což se dá udělat podle dat získaných od společnosti Gartner a Forrester. Tyto společnosti dělají analýzy trhu pro velké společnosti, které chtějí investovat do IT a také jsou hodně známé podle svých metod analýzy „Gartner Magic Quadrant“ a „Forrester Wave“. Důvodem, proč bylo zvoleno hodnocení v sítích datových center, je to, že SDN se právě v datových centrech nejvíc používají.

K vyhodnocení čtyř společností je potřeba zvolit kritéria, podle kterých se dají hodnotit. Pokud se jedná o datová centra, nejvhodnější kritéria: programovatelnost, sledování a viditelnost, operační systém a poskytovatelé hardwarů pro datová centra.

Data pro sledování a viditelnost, operační systém a programovatelnost se dá získat od společnosti Forrester. Nejpopulárnější poskytovatele hardwarů pro datová centra lze pak získat od společnosti Gartner.

Tab. 3.1: Data od společnosti Forrester

Společnost	Sledování a viditelnost	Programovatelnost	Operační systém
Cisco	4.40	3.70	1.70
Arista Networks	3.80	4.20	3.50
Huawei	3.60	4.10	2.90
Juniper Networks	3.80	4.30	3.80

Podle společnosti Gartner a její analýzy „Magic Quadrant for Data Center Networking“ lze vyhodnotit jako čtyři nejlepší společnosti Cisco, Arista Networks, Huawei a Juniper Networks. Hlavním kritériem hodnocení byla možnost aktuální

nabídky hardwaru, totiž ty, které jsou teď na trhu od společnosti a jsou už realizovány. Dokonce lze říct, že společnost Cisco a Arista Network jsou téměř na stejné úrovni [14].

Následně se dají od společnosti Forrester získat data pro programovatelnost, sledování a viditelnost, operační systém. Budou to bodovaná data, u nichž bude platit, že čím více bodů, tím lépe, viz tabulka 3.1 [15].

Z výsledků vyplývá, že Juniper Networks a Arista Networks se dělí o první a druhou pozici a mají podobné výsledky. Třetí pozici obsadila společnost Huawei a poslední Cisco. Pokud bychom ovšem sledovali více kritérií, byly by výsledky pravděpodobně jiné.

3.3 Budoucí trendy SDN

Většina SDN se používá v této fázi v datových centrech. Další vývojové trendy SDN by se mohly vytvořit, pokud by SDN bylo používáno i mimo datová centra. Dnes lze říct přesně pouze to, že SDN a NFV (Network Function Virtualization) budou čím dál tím více využívány společně. Ve světě reálného využívání již o tom existují důkazy:

- SD-WAN (Software Defined Networking in a Wide Area Network) — softwarově definované sítě WAN (Wide Area Networking) jsou využitím softwarové platformy pro správu k řízení přístupu ke vzdáleným či pobočkovým pracovištím organizace. IDC očekává, že se SD-WAN stane do roku 2020 trhem s obratem 5.25 miliard dolarů [16].
- Mikrosegmentace – stále více nasazení SDN se používá nejen k implementaci bezpečnostních produktů založených na softwaru, ale také k implementaci mikrosegmentace.
- Správa IoT (Internet of Things) — rostoucí počet připojených zařízení způsobuje příval síťových přenosů. Přívrženci SDN tvrdí, že softwarově založená vrstva správy sítě může pomáhat se stanovením priorit síťových přenosů a vykonávat analýzy typů přenosů v síti.
- SDN/NFV v 5G (Fifth Generation) – architektura 5G bude mít cloudový přístup, transport a základní sítě. Technologie virtualizace NFV a centralizované řízení SDN jsou integrovány do systému DNA 5G.

4 Praktická část – laboratorní úlohy

Hlavním účelem této diplomové práce bylo navrhnout laboratorní úlohy. Laboratorní úlohy mají sloužit k seznámení studentů s programovatelností API v SDN. V teoretické části bylo popsáno, co jsou RESR API a nástroj Ansible.

4.1 Laboratorní úloha 1 – konfigurace BIG-IP loadbalanceru pro vyrovnání zátěže serverů

Cílem laboratorní práce je seznámit studenty s možnostmi programování SDN API. Úloha se zabývá programováním SDN API pomocí skriptu, který je napsán v jazyce Python a pro komunikaci se serverem využívá rozhraní REST API. Následně je také vyzkoušena možnost konfigurování pomocí nástroje Ansible.

V laboratorní úloze bude obsaženo seznámí s Python a jak ho využít spolu s REST API i knihovnou F5-SDK (Software Development Kit). Pomocí Python skriptu a REST API bude nakonfigurován server pro vyrovnávání zátěže koncových serverů. Pro nácvik programovatelnosti SDN je využito hardwarové řešení BIG-IP od společnosti F5 Networks. Dalším bodem úlohy je využití nástroje Ansible pro stejný účel, jak tomu je v předchozí úloze a porovnání těchto metod konfigurování.

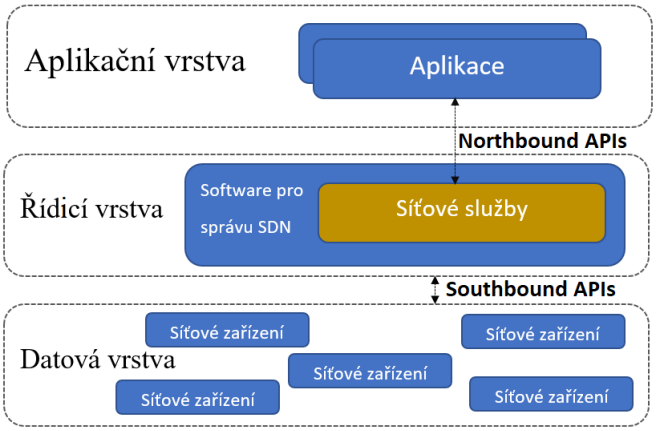
4.1.1 Zadání úlohy

1. Konfigurace BIG-IP loadbalanceru pro vyrovnávání zátěže serverů pomocí webového rozhraní BIG-IP.
2. Konfigurace BIG-IP loadbalanceru pro vyrovnávání zátěže serverů pomocí Python skriptu přes REST API.
3. Konfigurace BIG-IP loadbalanceru pro vyrovnávání zátěže serverů pomocí nástroje Ansible.

4.1.2 Teoretický úvod

SDN je nová síťová architektura, ve které je vrstva řízení sítě oddělena od zařízení pro přenos dat a je implementována programově. Architektura SDN má tři úrovně viz Obr. 4.1:

- Úroveň infrastruktury (datová vrstva), včetně sady síťových zařízení (přepínače a směrovače).
- Řídicí vrstva, která zahrnuje síťový operační systém, který poskytuje aplikacím síťové služby a softwarové rozhraní pro správu síťových zařízení a sítě.
- Aplikační vrstva pro flexibilní a efektivní správu sítě.



Obr. 4.1: Architektura SDN pro laboratorní úlohu číslo 1.

Architekturu SDN se dá rozdělit do dvou hlavních oblastí výměny informací. První je mezi aplikacemi SDN a druhá je pro správu fyzických síťových zařízení. První proud byl nazýván „Northbound“ rozhraní a druhý „Southbound“ rozhraní.

Interakce řídicí roviny a datové roviny se provádí pomocí „Southbound“ rozhraní. Nejznámějším protokolem pro interakci je OpenFlow, ale kromě toho existují i další: ForCES, OVSDB, POF OpFlex, OpenState, ROFL, HAL, PAD [2].

Interakce řídicí roviny a aplikační roviny se provádí pomocí „Northbound“ rozhraní. Každý kontrolér má obvykle své „Northbound“ rozhraní, protože standardizace pro něj dosud nebyla vyvinuta. Například kontroléry jako Floodlight, Trema, NOX, Onix a OpenDaylight používají vlastní „Northbound“ APIs [2].

RESTful API

Pro služby postavené s ohledem na REST, totiž ty, které neporušují jimi stanovená omezení, se používá termín RESTful. RESTful APIs se v síťovém průmyslu používají stále častěji, přestože se objevily až na začátku roku 2000. Většina API, která dnes existují v síťové infrastruktuře, jsou HTTP RESTful. Pokud se jedná o rozhraní RESTful API v síťovém zařízení nebo v kontroléru SDN, znamená to, že takové API poskytuje výměnu informací mezi klientem a serverem. Klient je obvykle reprezentován aplikací, jako je skript Pythonu nebo webová aplikace s uživatelským rozhraním, a server je síťové zařízení nebo kontrolér. Protože protokol HTTP se používá jako přenosový protokol, musí uživatel provádět některé operace s adresou URL, stejně jako při surfování po internetu.

Nástroj pro nastavení Ansible

Ansible – má decentralizovanou architekturu bez použití agentů a používá SSH jako základní přenosový protokol. Obvykle funguje na základě modelu „push“, ale také podporuje model „pull“. Nástroj Ansible je napsán v jazyce Python a používá tento jazyk k rozšíření funkčnosti. Obsahuje podporu práce se šablonami napsanými v jazyce Jinja. Ansible byl původně využíván jako prostředek k rychlému provádění specializovaných příkazů na serverech, ale postupem času se z něj vyvinul výkonný nástroj pro orchestraci úkolů pomocí takzvaných „knih“, které provádějí typické úkoly s konstantními výsledky na cílových systémech. Knihy skriptů mohou být psány ve standardním jazyce YAML nebo v dialektu jazyka YAML specializovaného na Ansible [11].

F5 BIG-IP

F5 definuje SDN jako architekturu pro navrhování sítí, která snižuje provozní náklady centralizací řízení do samostatné řídicí roviny. Která programově konfiguruje a rozšiřuje všechny prvky a služby datových cest prostřednictvím otevřeného API.

Protože se diskuse o SDN začaly zabývat výrobními nasazeními zahrnujícími aplikace. Dialog se přirozeně rozšířil, aby zahrnoval přímou integraci služeb vyšší vrstvy od 4 a dále. F5 dodává základní principy architektury SDN jako společné funkce (iControl, iRules) v celém svém portfoliu produktů od roku 2001.

GNU nano

Gnu nano je textový editor pomocí, kterého se dá vytvářet skripty a editovat textové konfigurace. Většina příkazů editoru GNU nano jsou založena na kombinaci Ctrl+znak. Důležité taktéž je, že se jedná o nemodální textový editor, což znamená, že jak editace textu, tak i zadávání příkazů se provádí v jediném režimu bez nutnosti jejich přepínaní, jako je tomu u editorů „vi“ a „Vim“.

4.1.3 Úkoly

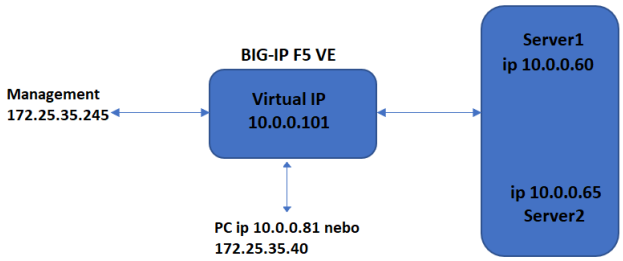
Prvním úkolem je konfigurace BIG-IP loadbalanceru pro vyrovnávání zátěže serverů přes webové rozhraní BIG-IP. Konfigurace pomocí webového rozhraní je využita pouze kvůli pochopení toho, co vše je třeba nastavit, protože další úkoly již nevyužívají toto uživatelské rozhraní.

Druhým úkolem je stejná konfigurace, ale pomocí REST API, Python skriptu a knihovny F5-SDK. Pro zjednodušení psaní skriptu je využito programovací prostředí PyCharm, ve kterém se dá jednoduše nainstalovat knihovnu F5-SDK.

I poslední úkol obsahuje stejnou konfigurace, avšak využit je nástroj Ansible, který je zprovozněn na konfiguračním serveru využitém k tomuto účelu.

4.1.4 Schéma zapojení

Zapojení laboratorní úlohy – přehled Obr. 4.2



Obr. 4.2: Zapojení laboratorní úlohy 1 – přehled.

4.1.5 Vybavení pracoviště

- 1 x stolní počítač.
- 1 x F5 Network BIG-IP.
- 1 x obraz disku virtuálního serveru LAB_Ansible_Nginx, který obsahuje OS Debian 10 bez GUI, Nginx, Ansible.
- 1 x obraz disku virtuálního serveru LAB_Nginx, který obsahuje OS Debian 10 bez GUI, Nginx.

4.1.6 Pracovní postup

1. **Konfigurování pomocí webového rozháraní**
   (a) BIG-IP: výchozí nastavení a připojení
   Rozhraní pro nastavení BIG-IP je na adrese `https://172.25.35.245`. Je potřeba provést obnovu konfigurace zařízení, pro to lze použít uloženou konfiguraci **pocatecni_konfigurace_LAB1.ucs**, pokud není jednotka ve výchozím stavu. To lze poznat podle toho, že v záložce **Local Traffic : Network Map** není žádný viditelný předkonfigurovaný server. Obnova konfigurace nějakou chvíli trvá a je možné, že bude vyžadován restart systému. Výchozí soubor konfigurace je v záložce **System ›› Archives**. Obnova se provede rozkliknutím archivu a zvolením tlačítka **Restore**. Konfiguraci lze rozdělit do jednotlivých kroků:

- Nodes (dále jen uzly) je konfigurován v záložce **Local Traffic-Nodes : Node List**.
- Pools je konfigurován v záložce **Local Traffic ›› Pools : Pool List**.
- Dalším krokem je konfigurace ip adresy virtuálního serveru **Local Traffic ›› Virtual Servers : Virtual Server List**.

Uzel je logická entita v BIG-IP, která identifikuje IP adresu fyzického zdroje v síti. Je možné explicitně vytvořit uzel nebo dát systému BIG-IP příkaz k jeho automatickému vytvoření, pokud je přidán člen poolu do poolu. Rozdíl mezi uzlem a členem poolu je v tom, že uzlu je přiřazena pouze IP adresa zařízení a člen poolu zahrnuje IP adresu a službu, například 10.0.10:80.

Pools je skupina, která obsahuje jeden nebo více uzlů, serverů, které zpracovávají aplikační provoz. Nejběžnějším typem serverového poolu jsou webové servery.

Virtuální server je jednou z nejdůležitějších součástí jakékoli konfigurace systému BIG-IP. Virtuální server je objekt řízení provozu v systému BIG-IP, který je reprezentován virtuální IP adresou a službou, například 192.168.20.10:80. Když klienti v externí síti odesílají přenos aplikací na virtuální server, virtuální server poslouchá tento přenos a přes překlad cílových adres směruje provoz v souladu s metodou nastavení parametrů na virtuálním serveru. Hlavním účelem virtuálního serveru je distribuovat provoz mezi poolem serveru, který je zvolen v konfiguraci virtuálního serveru.

(b) Pro konfiguraci uzlu lze využít záložku **Local Traffic ›› Nodes : Node List** Tab. 4.1.

Tab. 4.1: Konfigurace uzlů

Položka	Konfigurační informace
Name	10.0.0.60
Address	10.0.0.60
Položka	Konfigurační informace
Name	10.0.0.65
Address	10.0.0.65

(c) Pro konfiguraci poolu je možné použít záložku **Local Traffic ›› Pools : Pool List** Tab. 4.2.

(d) Pro konfiguraci Virtual Servers se dá použít záložka **Local Traffic ›› Virtual Servers : Virtual Server List** Tab. 4.3.

Tab. 4.2: Konfigurace pools

Položka	Konfigurační informace
Name	Pools_for_web
Health Monitors	http
New Members	Node list
	10.0.0.60:80
	10.0.0.65:80

Tab. 4.3: Konfigurace virtuálního serveru

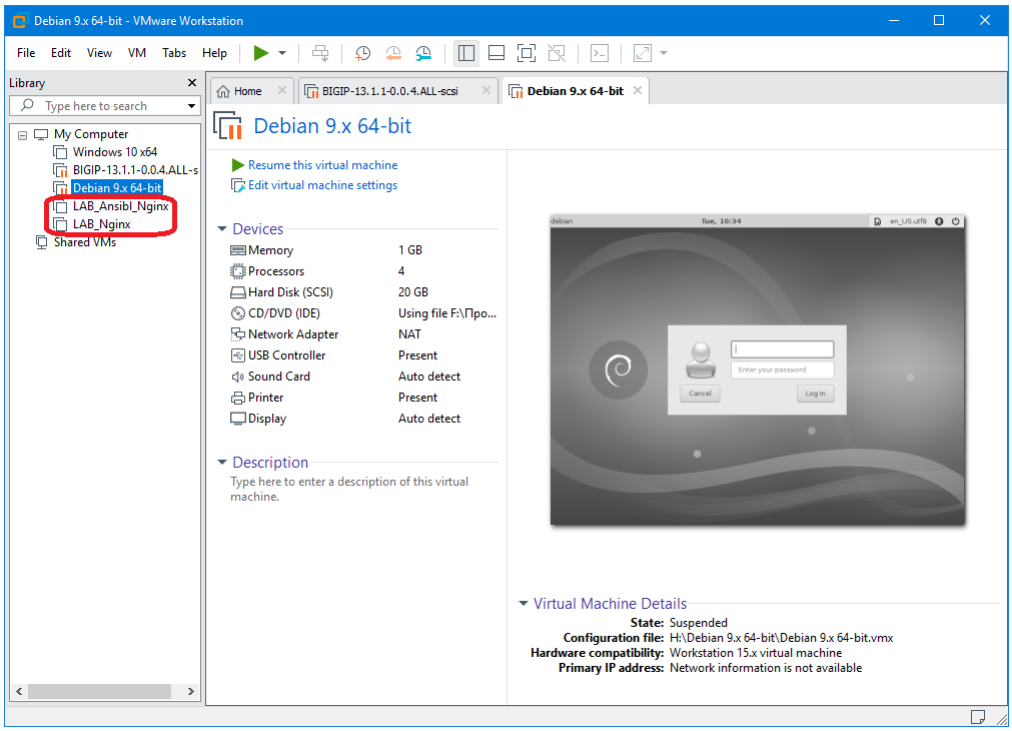
Položka	Konfigurační informace
Name	WEB_LB_ip
Destination_Address/Mask	10.0.0.70
Service_Port	80
Default Pool	Pools_for_web

(e) Zprovoznění serverů
Na ploše je program Wmplayer, ve kterém jsou dva servery. První server pojmenován LAB_Ansibl_Nginx a druhý LAB_Nginx viz Obr. 4.3. K serverům se lze připojit pomocí SSH klientu PuTTy, který je také na ploše. IP adresy, hesla a úcty zmínění v tabulce 4.4. Funkčnost serverů lze ověřit pomocí příkazu **ping** nebo otevřít přes webový prohlížeč odkazy `http://10.0.0.60` a `http://10.0.0.65`.

Tab. 4.4: IP adresy serverů, hesla a úcty

Položka	Informace
Name	LAB_Ansibl_Nginx
IP	10.0.0.60
Učet	lab, root
Heslo	lab, root
Name	LAB_Nginx
IP	10.0.0.65
Učet	lab, root
Heslo	lab, root

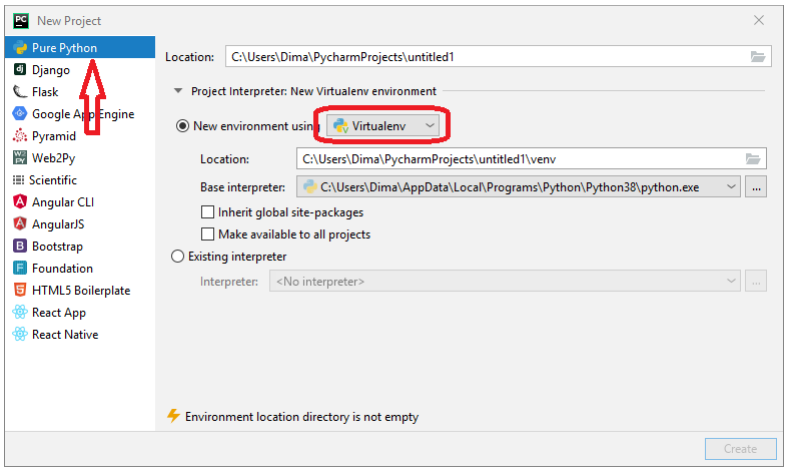
(f) Vyzkoušení serveru pro vyrovnávání zatížení
Z předchozí konfigurace je vidět, že server se nachází na ip adrese **10.0.0.70**. Jak přesně funguje server, lze vidět pomocí webového prohlížeče. Po přechodu na odkaz `http://10.0.0.70` je potřeba mačkat obnovení stránky.



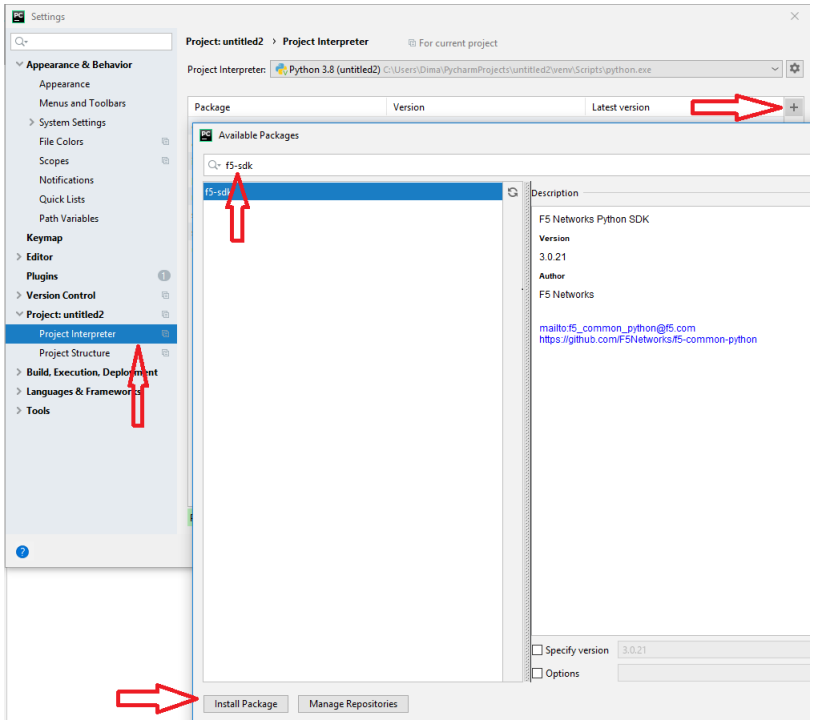
Obr. 4.3: Program Wmplayer.

Server začne pro vyrovnávání zatížení po každém připojení přesměrovávat uživatele mezi uzly. K tomu, proč byl konfigurován server přes webové rozhraní, je potřeba pochopit, co a jak konfigurovat v následujících bodech úlohy, kde není žádné webové rozháraní.

2. **Python a REST API**
   (a) Nyní lze přejít do nastavení serveru pro vyvažování zátěže přes Python a REST API. Je potřeba provést obnovu konfigurace BIG-IP, dále opakovat krok **1a** a následně vytvořit skript, který nakonfiguruje server pro vyrovnávání zatížení stejně, jak probíhalo konfigurování přes webové rozhraní.
   (b) Konfigurování bude probíhat pomocí Pythonu a F5-SDK. F5-SDK pro komunikace využívá RESTful API. První, co je potřeba udělat, je vytvořit v PyCharmu nový projekt **File » New Project** Obr. 4.4 a přidat do projektu paket F5-SDK **File » Settings** nebo **CTRL + ALT + S**, viz Obr. 4.5. Když bude nainstalován paket F5-SDK, lze vytvořit Python File **File » New » Python File** nebo **ALT + Insert(INS) » Python File** a pojmenovat LAB. Ověřit, že všechno funguje, se dá pomocí vyvolání balíčku F5-SDK v LAB.py řádkem **from f5.bigip import ManagementRoot**, když není hlášení o chybě, bylo vše uděláno správně.



Obr. 4.4: PyCharmu nový projekt.



Obr. 4.5: Přidání balíčku F5-SDK.

(c) Příklad konfigurování BIG-IP přes REST API

Z hlediska toho, že F5-SDK využívá pro komunikaci REST API, lze říct, že budou stejné kroky konfigurování, jako přes webové rozhraní, protože REST API využívá HTTP pro komunikaci.

Příklad konfigurování uzlu je vidět ve výpisu 4.1 [17].

```
from f5.bigip import ManagementRoot
# Connect to the BigIP zapnut debag aby pak ho vipisat
mgmt = ManagementRoot("192.168.1.100", "admin", "admin",
                      debug=True)
# Dostaneme info pro existujici nodes
nodes = mgmt.tm.ltm.nodes.get_collection()
for node in nodes:
           print(node.name)
# Pridame nodu
mynodes1 = mgmt.tm.ltm.nodes.node.create(
    name='10.0.0.60', address='10.0.0.60')

```

Výpis 4.1: Příklad konfigurování uzlu

Příklad odpovědi ve formátu JSON po konfigurování skriptem 4.1 vypadá následovně viz výpis 4.2.

```
 "kind": "tm:ltm:node:nodecollectionstate",
    "selfLink": "https://localhost/mgmt/tm/ltm/node?ver=13.1.1",
    "items": [
            "kind": "tm:ltm:node:nodestate",
            "name": "10.0.0.60",
            "partition": "Common",
            "fullPath": "/Common/10.0.0.60",
            "generation": 275,
            "selfLink": "https://localhost/mgmt/tm/ltm/node/~
                                Common~10.0.0.60?ver=13.1.1",
            "address": "10.0.0.60",
            "connectionLimit": 0,
            "dynamicRatio": 1,
            "ephemeral": "false",
            "fqdn":
                "addressFamily": "ipv4",
                "autopopulate": "enabled",
                "downInterval": 5,
                "interval": "3600"

```

Výpis 4.2: Příklad odpovědi ve formátu JSON pro uzel

Z výpisu 4.2 je vidět, že vše, co je potřeba pro konfiguraci a také odkazy, je možné najít v odpovědích JSON. Dále budou výpisy JSON pro pool výpis 4.3, přidání uzlů do poolu výpis 4.4 a virtuální server výpis 4.5. Je potřeba dávat pozor na povinné položky při konfiguraci.

(d) Po dokončení skriptu je potřeba ho ukázat a popsat, co dělá.

```
    "kind": "tm:ltm:pool:poolcollectionstate",
    "selfLink": "https://localhost/mgmt/tm/ltm/pool?ver=13.1.1",
    "items": [
            "kind": "tm:ltm:pool:poolstate",
            "name": "pools_for_WEB",
            "partition": "Common",
            "fullPath": "/Common/pools_for_WEB",
            "generation": 282,
            "selfLink": "https://localhost/mgmt/tm/ltm/pool/
                ‘           ~Common~pools_for_WEB?ver=13.1.1",
            "allowNat": "yes",
            "allowSnat": "yes",
            "ignorePersistedWeight": "disabled",
            "ipTosToClient": "pass-through",
            "ipTosToServer": "pass-through",
            "linkQosToClient": "pass-through",
            "linkQosToServer": "pass-through",
            "loadBalancingMode": "round-robin",
            "minActiveMembers": 0,
            "minUpMembers": 0,
            "minUpMembersAction": "failover",
            "minUpMembersChecking": "disabled",
            "monitor": "/Common/http ",

```


Výpis 4.3: Příklad odpovědi ve formátu JSON pro pool


```
"kind": "tm:ltm:pool:members:memberscollectionstate",
    "selfLink": "https://localhost/mgmt/tm/ltm/pool/pools_for_WEB/members?ver=13.1.1
     ",
    "items": [
            "kind": "tm:ltm:pool:members:membersstate",
            "name": "10.0.0.60:80",
            "partition": "Common",
            "fullPath": "/Common/10.0.0.60:80",
            "generation": 277,
            "selfLink": "https://localhost/mgmt/tm/ltm/pool/pools_for_WEB/members
                        /~Common~10.0.0.60:80?ver=13.1.1",
            "address": "10.0.0.60",
            "connectionLimit": 0,
            "dynamicRatio": 1,
            "ephemeral": "false",
            "fqdn": {
                "autopopulate": "enabled"
            },
            "inheritProfile": "enabled",
            "logging": "disabled",
            "monitor": "default",
            "priorityGroup": 0,
            "rateLimit": "disabled",
            "ratio": 1,
            "session": "monitor-enabled",
            "state": "down"

```


Výpis 4.4: Příklad odpovědi ve formátu JSON pro přidání uzlu do pool

```
    "kind": "tm:ltm:virtual:virtualcollectionstate",
    "selfLink": "https://localhost/mgmt/tm/ltm/virtual?ver=13.1.1",
    "items": [
            "kind": "tm:ltm:virtual:virtualstate",
            "name": "WEB_LB_ip",
            "partition": "Common",
            "fullPath": "/Common/WEB_LB_ip",
            "generation": 283,
            "selfLink": "https://localhost/mgmt/tm/ltm/virtual/~
                        Common~WEB_LB_ip?ver=13.1.1",
            "addressStatus": "yes",
            "autoLasthop": "default",
            "cmpEnabled": "yes",
            "connectionLimit": 0,
            "destination": "/Common/10.0.0.70:80",
            "enabled": true,
            "gtmScore": 0,
            "ipProtocol": "tcp",
            "mask": "255.255.255.255",
            "mirror": "disabled",
            "mobileAppTunnel": "disabled",
            "nat64": "disabled",
            "pool": "/Common/pools_for_WEB",

```


Výpis 4.5: Odpověď ve formátu JSON pro přidání virtuálního serveru

3. **Ansible**
   (a) Nyní lze přejít do nastavení serveru pro vyvažování zátěže přes Python a REST API. Je potřeba provést obnovu konfigurace BIG-IP, dále opakovat krok 1a a následně vytvořit skript, který nakonfiguruje server pro vyrovnávání zatížení stejně, jako probíhalo konfigurování přes webové rozhraní.
   (b) Na prvním serveru, který je pojmenován LAB_Ansibl_Nginx, už je nainstalován Ansible verze 2.9. Nejlepší cesta pro konfigurování je připojit se pomocí SSH do serveru. V této situaci SSH klient PuTTY. Cesta k hlavnímu souboru Ansible je **/etc/ansible/**, kde je potřeba vytvořit programovací skript pomocí příkazu **nano**, koncovka složky má být **.yml**, například **nano /etc/ansible/LAB.yml**. Ansible využívá YAML Syntax, proto je potřeba respektovat strukturu textu pro YAML. Příkazem **ansible-playbook LAB.yml** je možné spustit skript. Funguje pouze pokud se nachází ve složce **/etc/ansible/** [18].
   (c) Příklad konfigurování BIG-IP přes Ansible
   Příklad konfigurování uzlu je vidět ve výpisu 4.6. Ověřit, zda je skript funkční, lze přes webové rozhraní. Pro konfigurování poolu je potřeba využít modul **bigip_pool**. Pro přidání uzlu do poolu lze využít **bigip_pool_member** a pro vytvoření virtuálního serveru **bigip_virtual-**

_server [19].

```
---

- name: An example copy playbook
  hosts: all
  connection: local
  vars:
    server1: "10.0.0.60"
  tasks:
   - name: Add node Server1
     bigip_node:
      host: "{{ server1 }}"
      name: "{{ server1 }}"
      provider:
       server: 172.25.35.245
       user: admin
       password: admin
       validate_certs: False
     delegate_to: localhost

```

Výpis 4.6: Příklad konfigurování uzlu přes Ansible

Více o proměnných, které využívají moduly, lze vidět v tabulce 4.5.

Tab. 4.5: Proměnné pro moduly BIG-IP

bigip_pool	bigip_pool_member	bigip_virtual_server
description	address	destination
monitors	description	name
name	name	pool
provider	pool	port
	port	provider
	provider	

(d) Po dokončení skriptu je potřeba ho ukázat a popsat, co dělá.

4.1.7 Kontrolní otázky

- Jak se jmenuje metoda, pomocí které lze získat stav existujícího zdroje v F5-SDK?
- Co využívá Ansible pro komunikaci se serverem BIG-IP?
- Ve které vrstvě referenčního modelu OSI (Open Systems Interconnection) funguje server pro vyrovnávání zatížení?
- Je možné nakonfigurovat v BIG-IP server pro vyrovnávání zatížení na druhé vrstvě referenčního modelu OSI?

4.1.8 Úklid pracoviště

- Smazat skripty, které byly během úlohy vytvořeny.
- Vypnout virtuální stanice v Wmplayeru.

4.2 Laboratorní úloha 1 – pokyny pro vyučující

- **Skript pro python**

```
1  from f5.bigip import ManagementRoot
2  # Pripojeni do BigIP, zapnut debag aby pak ho vipisat
3  mgmt = ManagementRoot("172.25.35.245", "admin", "admin",
4                        debug=True)
5  # Dostanemo info pro existujici nodes
6  nodes = mgmt.tm.ltm.nodes.get_collection()
7  for node in nodes:
8             print(node.name)
9  # Vytvorime nodes
10 mynodes1 = mgmt.tm.ltm.nodes.node.create(
11     name='Server1', address='10.0.0.60')
12 mynodes2 = mgmt.tm.ltm.nodes.node.create(
13     name='Server2', address='10.0.0.65')
14 # Vytvorime pools a pridame nodes do poolu
15 mypool1 = mgmt.tm.ltm.pools.pool.create(
16     name='pools_for_web', monitor='/Common/http')
17 members1 = mypool1.members_s.members.create(
18     partition='Common', name='Server1:80')
19 members2 = mypool1.members_s.members.create(
20     partition='Common', name='Server2:80')
21 #Overime, co jsme pridali
22 pools = mgmt.tm.ltm.pools.get_collection()
23 print("---------------------------")
24 print("Pool names:")
25 for pool in pools:
26   print(pool.name)
27 print("---------------------------")
28 print("Members names pool name='pools_for_web':")
29 members = mypool1.members_s.get_collection()
30 member = mypool1.members_s.members
31 for member in members:
32     print(member.name)
33 # Vytvorime virtual server
34 server1 = mgmt.tm.ltm.virtuals.virtual.create(
35     name='WEB_LB_ip',
36     destination='/Common/10.0.0.70:80', pool='pools_for_web')
37 # Overime, ze jsme vytvorili server
38 servers = mgmt.tm.ltm.virtuals.get_collection()
39 print("---------------------------")
40 print("Virual servers name:")
41 for server in servers:
42     print(server.name)
43 print("-----------END-------------")
```

Výpis 4.7: Příklad skriptu v jazyce Python

- **Skript pro Ansible**

  JProblém nastane, když je potřeba přidat existující uzel s tím, že se jeho jméno liší od ip adresy. Pokud by se měl přidat takový uzel, kde se ip adresa liší od jména uzlu, je potřeba vytvořit uzel přímo v modulu **bigip_pool_member**, jinak vždy vznikne chyba pro duplicitu.

```
---
- name: An example copy playbook
  hosts: all
  connection: local
  vars:
    server1: 10.0.0.60
    server2: 10.0.0.65
  tasks:
   - name: Add nodes
     bigip_node:
      host: "{{ item.serv }}"
      name: "{{ item.serv }}"
      provider:
       server: 172.25.35.245
       user: admin
       password: admin
       validate_certs: False
     delegate_to: localhost
     loop:
         - { serv: "{{ server1 }}" }
         - { serv: "{{ server2 }}" }
   - name: Create a pool
     bigip_pool:
      name: pools_for_WEB
      lb_method: round-robin
      monitors:
      - http
      provider:
       server: 172.25.35.245
       user: admin
       password: admin
       validate_certs: False
     delegate_to: localhost
   - name: Add nodes to pool
     bigip_pool_member:
            host: "{{ item.host }}"
            port: "80"
            pool: "pools_for_WEB"
            state: "present"
            provider:
                server: 172.25.35.245
                user: admin
                password: admin
                validate_certs: False
     loop:
            - { host: "{{ server1 }}" }
            - { host: "{{ server2 }}" }
     delegate_to: localhost
   - name: Create WEB_LB_ server
     bigip_virtual_server:
```

```
51        destination: 10.0.0.70
52        name: WEB_LB_ip
53        pool: pools_for_WEB
54        port: "80"
55        provider:
56         server: 172.25.35.245
57         user: admin
58         password: admin
59         validate_certs: False
60       delegate_to: localhost  }
```

Výpis 4.8: Příklad skriptu pro Ansible

4.2.1 Odpovědi na kontrolní otázky

- Jak se jmenuje metoda, pomocí které lze získat stav existujícího zdroje v F5-SDK? (get_collection nebo load)
- Co využívá Ansible pro komunikaci se serverem BIG-IP? (REST API,HTTP)
- Ve které vrstvě referenčního modelu OSI funguje server pro vyrovnávání zatížení? (7 vrstva)
- Je možné nakonfigurovat v BIG-IP server pro vyrovnávání zatížení na druhé vrstvě referenčního modelu OSI? (Ano)

4.3 Laboratorní úloha 2 – konfigurace „Leaf-Spine“ Layer 3/ECMP sítě

Cílem laboratorní práce je seznámit studenty s možnostmi programování SDN API. Úloha se zabývá programováním SDN API pomocí skriptu, který je napsán v jazyce Python a využívá pro komunikaci s routery rozhraní eAPI. Následně je také vyzkoušena možnost konfigurování pomocí nástroje Ansible. V laboratorní úloze bude seznámení s Python a jak ho využít spolu s eAPI i knihovnou „pyeapi“. Pomocí Python skriptu a eAPI bude nakonfigurovaná jednoduchá síť „Clos“, také známá pod jménem „Leaf-Spine“. Pro nácvik programovatelnosti SDN je využita Arista vEOS. Dalším bodem úlohy je využití nástroje Ansible pro stejný účel, jak tomu je v předchozí úloze a porovnaní těchto metod konfigurování.

4.3.1 Zadání úlohy

1. Konfigurace jednoduché sítě „Leaf-Spine“ Layer 3/ECMP (Equal-Cost Multi-Path Routing) a VXLAN (Virtual Extensible Local Area Network) pomocí Python skriptu přes eAPI.

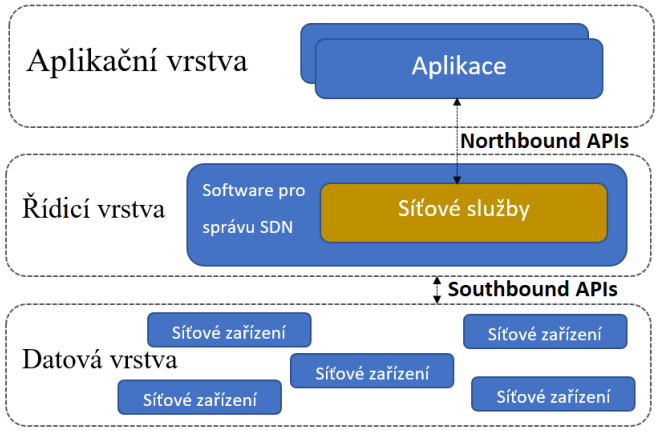
2. Konfigurace jednoduché sítě „Leaf-Spine“ Layer 3/ECMP a VXLAN pomocí nástroje Ansible.

4.3.2 Teoretický úvod

SDN

SDN je nová síťová architektura, ve které je vrstva řízení sítě oddělena od zařízení pro přenos dat a je implementována programově. Architektura SDN má tři úrovně viz Obr. 4.6:

- Úroveň infrastruktury (datová vrstva), včetně sady síťových zařízení: přepínače a směrovače.
- Řídicí vrstva, která zahrnuje síťový operační systém, který poskytuje aplikacím síťové služby a softwarové rozhraní pro správu síťových zařízení a sítě.
- Aplikační vrstva pro flexibilní a efektivní správu sítě.



Obr. 4.6: Architektura SDN pro laboratorní úlohu číslo 2.

Architektura SDN se dá rozdělit do dvou hlavních oblastí výměny informací. První je mezi aplikacemi SDN a druhá je pro správu fyzických síťových zařízení. První proud byl nazýván „Northbound“ rozhraní a druhý „Southbound“ rozhraní.

Interakce řídicí roviny a datové roviny se provádí pomocí „Southbound“ rozhraní. Nejznámějším protokolem pro interakci je OpenFlow, ale kromě toho existují i další: ForCES, OVSDB, POF OpFlex, OpenState, ROFL, HAL, PAD [2].

Interakce řídicí roviny a aplikační roviny se provádí pomocí „Northbound“ rozhraní. Každý kontrolér má obvykle své „Northbound“ rozhraní, protože standardizace pro „Northbound“ rozhraní dosud nebyla vyvinuta. Například kontroléry jako Floodlight, Trema, NOX, Onix a OpenDaylight používají vlastní „Northbound“ APIs [2].

ARISTA EOS

EOS – je modulární operační systém, který poskytuje Arista Network. Je to jeden operační systém pro celou řadu přepínačů.

EOS je založen na Fedora OS. Operační systém běží na samostatném procesoru, který umožňuje oddělit řídicí rovinu - CPU (Central Processing Unit), EOS a datovou rovinu „rovina přenosu dat“ - ASIC (Application-specific Integrated Circuit). To vše není nic nového, ale v EOS existují architektonické prvky, které nejsou v OS jiných dodavatelů. Například komponenty potřebné pro fungování přepínače si navzájem přímo nevyměňují data, ale dělají to pouze prostřednictvím speciální manažerské základny – Sysdb. Sysdb je jak společná sběrnice pro komunikaci mezi procesy, tak databáze pro pracovní informace o těchto procesech.

vEOS-lab - rozšiřuje stejnou platformu EOS, totiž je to omezený EOS, který se dá spustit pomocí hypervizoru. Zákazníci mohou používat vEOS-lab pro návrh sítě, rychlý vývoj a testování EOS a jako nástroj pro vzdělávací iniciativy.

EOS API (eAPI) - umožňuje aplikacím a skriptům mít plnou programovou kontrolu nad EOS. eAPI využívá datový formát JSON přes HTTP. Ve většině případů budou předány CLI příkazy, ale existuji i API moduly, které se dají vyvolat, například BGP, MLAG (Multi-Chassis Link Aggregation), OSPF (Open Shortest Path First) atd.

Nástroj pro nastavení Ansible

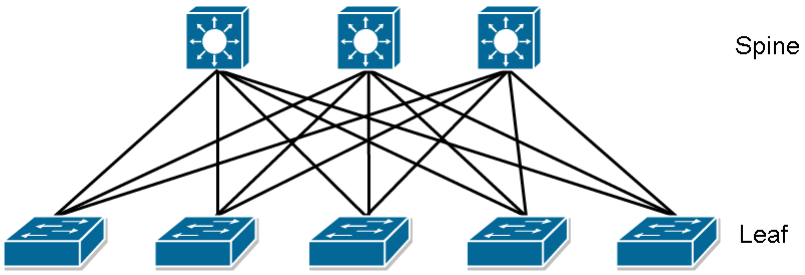
Ansible – má decentralizovanou architekturu bez použití agentů a používá SSH jako základní přenosový protokol. Obvykle funguje na základě modelu „push“, ale také podporuje model „pull“. Nástroj Ansible je napsán v jazyce Python a používá tento jazyk k rozšíření funkčnosti. Obsahuje podporu práce se šablonami napsanými v jazyce Jinja. Ansible byl původně využíván jako prostředek k rychlému provádění specializovaných příkazů na serverech, ale postupem času se z něj vyvinul výkonný nástroj pro orchestraci úkolů pomocí takzvaných „knih“, které provádějí typické úkoly s konstantními výsledky na cílových systémech. Knihy skriptů mohou být psány ve standardním jazyce YAML nebo v dialektu jazyka YAML specializovaného na Ansible [11].

EVE-NG

EVE-NG je síťový emulační software s více klientskými sítěmi, který poskytuje skvělé příležitosti profesionálům v oblasti sítí a zabezpečení. Umožňuje simulovat zařízení různých výrobců v jedné laboratoři, například Cisco, Juniper, Mikrotik, F5, Palo Alto, Huawei, vEOS-lab a td.

„Leaf-Spine“ síť

S konfigurací Leaf-Spine mají všechna zařízení přesně stejný počet segmentů a mají předvídatelné a konzistentní zpoždění informací. To je možné díky nové konstrukci topologie, která má pouze dvě vrstvy: vrstvy „Leaf“ a „Spine“. Vrstva „Leaf“ se skládá z přístupových přepínačů, které se připojují k zařízením, jako jsou servery, brány firewall, vyrovnávače zatížení a hraniční směrovače. Vrstva páteře, která se skládá ze směrovacích přepínačů, je páteří sítě, kde každý přepínač „Leaf“ je připojen ke každému přepínači „Spine“. Příklad takové sítě je vidět na 4.7.



Obr. 4.7: Model sítě „Leaf-Spine“.

GNU nano

Gnu nano – je textový editor, pomocí kterého se dají vytvářet skripty a editovat textové konfigurace. Většina příkazů editoru GNU nano je založena na kombinaci Ctrl+znak. Důležité taktéž je, že se jedná o nemodální textový editor, což znamená, že jak editace textu, tak i zadávání příkazů se provádí v jediném režimu bez nutnosti jejich přepínaní, jako je tomu u editorů „vi“ a „Vim“.

BGP

BGP – je hlavní dynamický směrovací protokol, který se používá na internetu.

Routery využívající BGP si vyměňují informace o dostupnosti sítě. Spolu s informacemi o sítích jsou přenášeny různé atributy těchto sítí, pomocí kterých BGP vybere nejlepší trasu a konfiguruje směrovací politiky.

Jedním z hlavních atributů přenášených s informacemi o trase je seznam AS (Autonomous System), kterými tato informace prošla. Tato informace umožňuje BGP určit, kde je síť umístěna vzhledem k autonomním systémům a vyloučit směrovací smyčky.

ECMP

ECMP – je mechanismus nebo dokonce strategie, když aby bylo možné doručit více paketů jednomu příjemci, tyto pakety neprochází jednou nejlepší cestou, ale procházejí několika nejlepšími cestami - obvykle určovány metrickými údaji.

Dynamické směrovací protokoly - OSPF, IS-IS (Intermediate System to Intermediate System), EIGRP (Enhanced Interior Gateway Routing Protocol), BGP - podporují ECMP. Maximální počet tras, po kterých bude provoz distribuován, závisí na implementaci - obvykle jsou to nejméně 4, ale software se zvyšuje do 32.

VXLAN

VXLAN (Virtual eXtensible Local Area Network) – je schéma enkapsulace vrstev druhého protokolu sítě ve protokolech třetí vrstvy (tunelování) za účelem vytvoření virtuální sítě a obejití některých omezení tradičního schématu VLAN. VXLAN běží na existující síťové infrastruktuře, takže se dá ji nazvat „overlay“ síť.

4.3.3 Úkoly

Prvním úkolem je konfigurace jednoduché sítě „Leaf-Spine“ Layer 3/ECMP a VXLAN pomocí Python skriptu přes eAPI. V EVE-NG je už vytvořena topologie sítě „Leaf-Spine“ podle všech pravidel, zbývá jenom napsat konfigurační skript v Python. Pro psaní skriptu bude využito prostředí PyCharm, ve kterém je potřeba nainstalovat knihovnu pyeapi. Topologie je vytvořena pomocí směrovačů od Arista Network. Konfigurovat EOS přes eAPI se dá dvěma směry. V prvním je využit modul, který bude předávat CLI příkazy. Konfigurace bude vypadat stejně jako základní konfigurace přes CLI. Druhý směr předpokládá využití modulů pro konfigurace, totiž pokud je potřeba konfigurovat BGP, je potřeba využít odpovídající modul. Modul uvnitř sebe také dovoluje využít CLI příkaz, jenomže příkaz bude věnován modulu, vyžít se totiž dají pouze CLI příkazy modulu. Lze tedy říct, že první úkol má hodně menších úkolů, které se týkají konfigurace IP adres, BGP, ECMP a Vxvlan. Více bude téma rozepsáno v postupu úlohy.

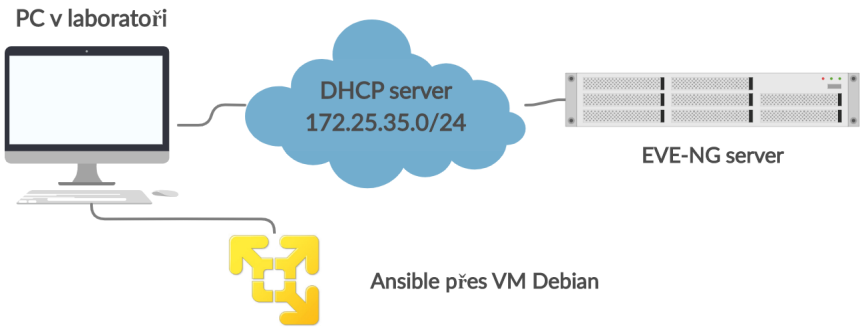
Poslední úkol obsahuje stejnou konfiguraci, avšak využit je nástroj Ansible, který je zprovozněn na konfiguračním serveru využitém k tomuto účelu. V Ansible se nedá využít CLI příkaz v modulu, ale je možné vyvolat modul přes SSH spojení. Jediný modul, který dovolí využít CLI příkazy, podporuje jenom jeden příkaz nebo více, ale stejného druhu. Pokud je potřeba poslat CLI příkazy, je nezbytné využít konfigurační soubor napsaný v jazyce Jinja a modul, který dovolí to udělat. Pro lepší pochopení programobility budou oba směry propojené.

Výsledkem konfigurování bude funkční síť, která využívá pro směrovaní BGP, a pomocí ECMP a VXLAN se dá poslat příkaz „ping“ mezi počítači, které se nachází v „overlay“ a „underlay“ síti.

Nezbytným faktorem pro obě úlohy je využití pro konfiguraci eAPI.

4.3.4 Schéma zapojení

Zapojení laboratorní úlohy – přehled 4.8. Víc o tom, jak vypadá topologie sítě, která je vytvořena v EVE-NG, lze vidět na obrázku 4.9.



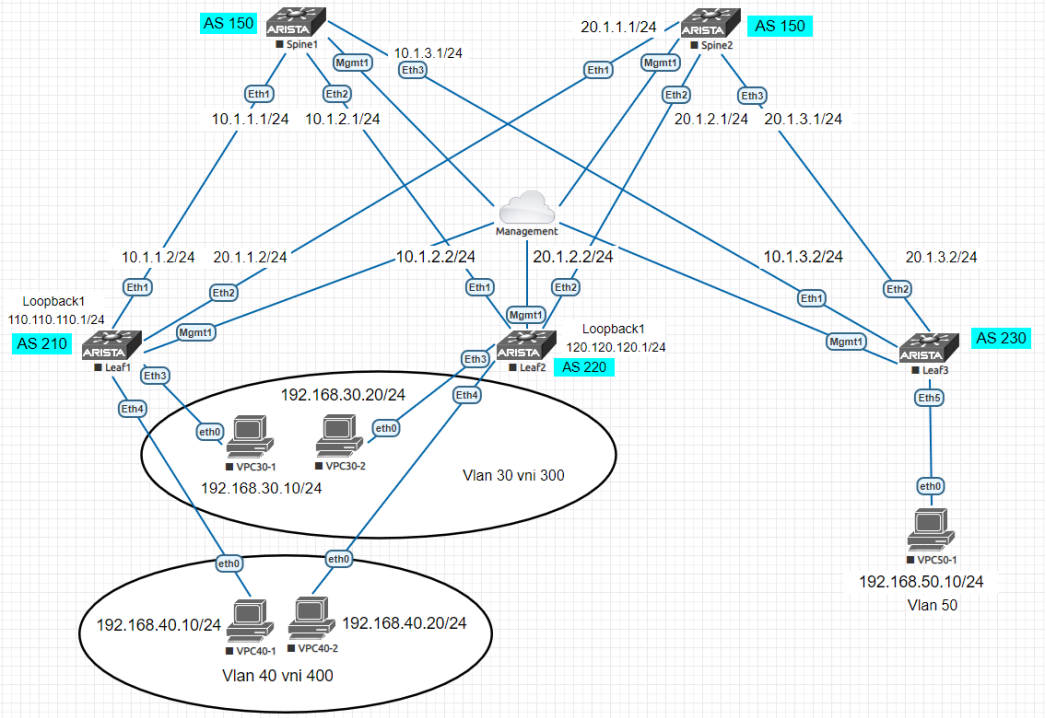
Obr. 4.8: Zapojení laboratorní úlohy 2 – přehled

4.3.5 Vybavení pracoviště

- 1 x stolní počítač.
- 1 x obraz disku virtuálního serveru LAB_Ansible, který obsahuje OS Debian 10 bez GUI a Ansible.
- 1 x EVE-NG server.

4.3.6 Pracovní postup

1. Konfigurování pomocí Python a eAPI
   (a) Otevřít ENE-NG, který se nachází na IP adrese `http://172.25.35.26/`, uživatelské jméno a heslo je **Lab**. Po přihlášení je potřeba otevřít záložku **Main » Arista LAB » LAB » Open**. Zleva se dá najít **Startup-configs**, ve kterém jsou výchozí konfigurace směrovačů a virtuálních počítačů. Konfigurace se liší pouze jmény hostů a IP adresami. Před spuštěním topologie je potřeba ověřit, že všech 5 směrovačů a VPC (Virtual Personal Computer) mají své výchozí konfigurace podle tabulky 4.6. Po-



Obr. 4.9: Topologie sítě „Leaf-Spine“.

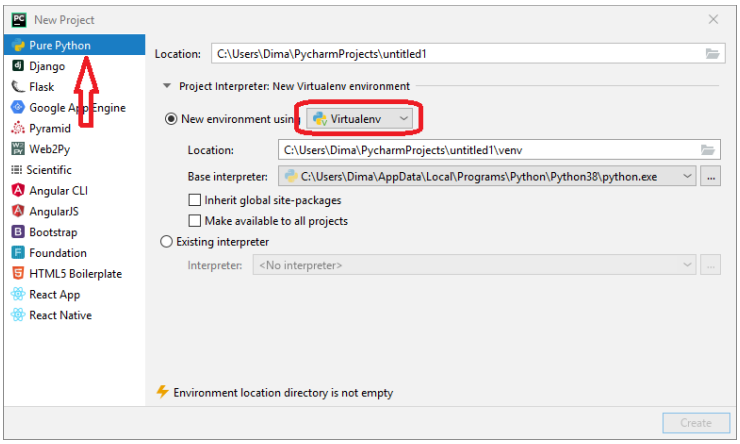
Tab. 4.6: Výchozí konfigurace

Host	IP adres /24	VPC	IP adres /24	Výchozí brána
Spine1	172.25.35.151	VPC30-1	192.168.30.10	192.168.30.254
Spine2	172.25.35.152	VPC30-2	192.168.30.20	192.168.30.254
Leaf1	172.25.35.161	VPC40-1	192.168.40.10	192.168.40.254
Leaf2	172.25.35.162	VPC40-2	192.168.40.20	192.168.40.254
Leaf3	172.25.35.163	VPC50-1	192.168.50.10	192.168.50.254

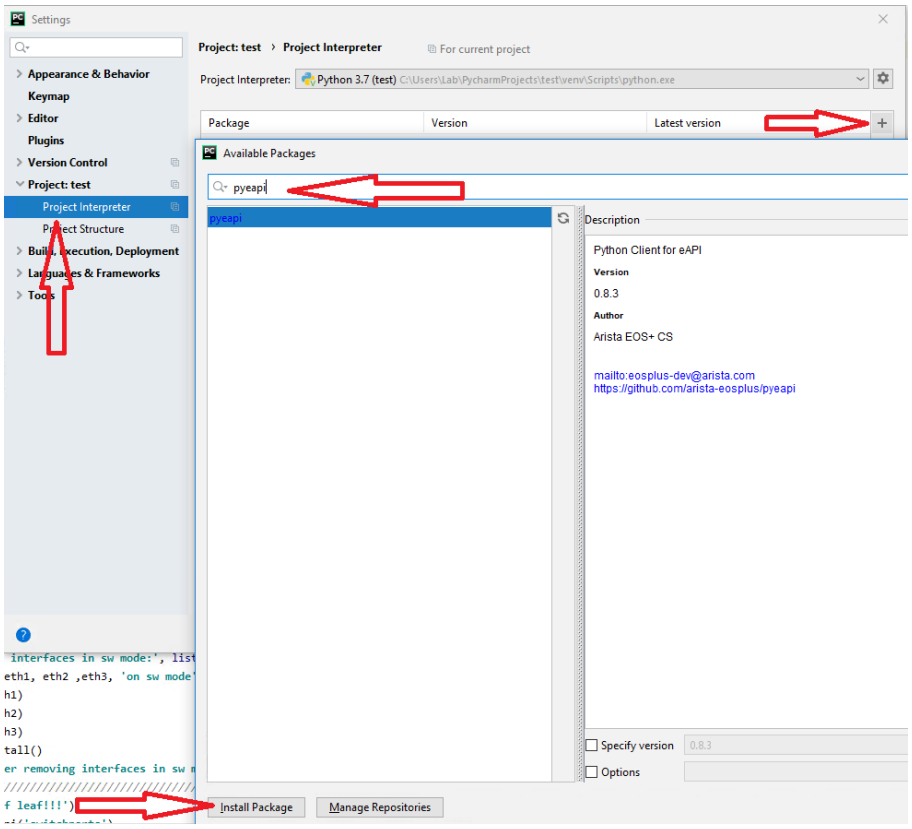
kud je všechno v pořádku, je potřeba rozkliknout **More actions » Wipe all nodes**, vyčkat dokud neproběhne vymazání předchozí konfigurace a pak rozkliknout **More actions » Start all nodes** a chvíli vyčkat. Všechna rozhraní řízení směrovačů jsou připojená do společného DHCP (Dynamic Host Configuration Protocol) serveru. Ne vždy může z prvního pokusu proběhnout vymazání předchozí konfigurace, proto ověřit, že se všechno podařilo lze pomocí příkazu **Ping IP adresy hostů**, IP adresy viz tabulka 4.6. Pokud se ručně rozklikne 1 host, dá se manuálně spustit, zastavit nebo vymazat předchozí konfigurace.

(b) konfigurování bude probíhat pomocí Pythonu a „pyeapi“ balíčku v prostředí PyCharm. První, co je potřeba udělat, je vytvořit v PyCharmu

nový projekt **File » New Project** Obr. 4.10 a přidat do projektu balíček „pyeapi“, **File » Settings** nebo **CTRL + ALT + S**, viz Obr. 4.11.



Obr. 4.10: PyCharmu nový projekt



Obr. 4.11: Přidání balíčku „pyeapi“

Když bude nainstalován balíček „pyeapi“, lze vytvořit Python File **File**

» New » Python File nebo **ALT + Insert(INS) » Python File** a pojmenovat ho LAB. Ověřit, že všechno funguje, se dá pomocí vyvolání balíčku „pyeapi“ v LAB.py řádkem **import pyeapi**, když se neukáže hlášení o chybě, bylo vše uděláno správně. Pak je potřeba vytvořit konfigurační soubor, ve kterém budou IP adresy a přihlašovací údaje. Soubor by měl vypadat následovně, viz výpis 4.9. Pomocí příkazu **pyeapi.load_config('host.conf')** se dají přenést data do balíčku „pyeapi“.

```
[connection:Spine1]
host: 172.25.35.151
transport: https
username: lab
password: lab
[connection:Spine2]
host: 172.25.35.152
transport: https
username: lab
password: lab
[connection:Leaf1]
host: 172.25.35.161
transport: https
username: lab
password: lab
[connection:Leaf2]
host: 172.25.35.162
transport: https
username: lab
password: lab
[connection:Leaf3]
host: 172.25.35.163
transport: https
username: lab
password: lab
```

Výpis 4.9: Příklad souboru host.conf

Nyní se dá navázat spojení se směrovači pomocí příkazů viz výpis 4.10. Spustit Python skript se dá pomocí **Run » Run**. Od této chvíle je potřeba pochopit, co je nutno nakonfigurovat. Pokud je řečeno, že se úloha týka Layer 3/ECMP a VXLAN, znamená to, že je potřeba propojit směrovače pomocí IP adres. Pak je nutné nakonfigurovat ECMP a následně VXLAN. Nejvíc pro tuto úlohu vyhovuje BGP. Logicky bude postup pro konfigurování vypadat takto, viz tabulka 4.7. Pro využití modulu je potřeba ho propojit s proměnou. Příklad propojení **swmode=Spine1.api('switchports')**, nyní se přes proměnu dá poslat příkaz, například **swmode.delete(Ethernet3)**. Pro přímé příkazy CLI není potřeba zadávat proměny, například **Spine1.run_commands(['enable', 'configure', 'ip routing'])**. ECMP se dá nastavit přímo v modulu BGP. Pro zjednodušení konfigurace BGP je potřeba využít **bgpc.configure_bgp('redist-**

ribute connected'). Dále v tabulkách: 4.9, 4.10, 4.11, 4.12, 4.8, 4.13 budou popsané možnosti modulů, které je potřeba využít. Pro zjednodušení konfigurace lze využít připravené tělo skriptu, které se nachází na ploše pracovního počítače v složce Arista LAB. V těle skriptu je potřeba měnit jenom položky, co jsou označeny **#pridat**. Tělo přesně opakuje tabulku 4.7 a postupně prochází směrovače. V EVE-NG jsou uvedená data jako IP adresy, AS BGP atd. Pro lepší přehled jsou konfigurační data také v tabulce 4.14, 4.15. Ve výpisech 4.12 a 4.12 jsou příklady jednoduchých skriptů.

(c) Při konfigurování BGP je nutno zadat číslo AS, IP adresu souseda a číslo AS, redistribuce přepojených sítě, „maximum-paths" se bude rovnat 2 a ECMP také. Pro VXLAN je potřeba zadat, jaké Vlan do kterého VNI (Virtual Network Identifier) přeložit, zdrojové rozhraní a VTEP (Vitual Tunnel End Point) [20].

```
1 Spine1 = pyeapi.connect_to('Spine1')
2 Spine2 = pyeapi.connect_to('Spine2')
3 Leaf1 = pyeapi.connect_to('Leaf1')
4 Leaf2 = pyeapi.connect_to('Leaf2')
5 Leaf3 = pyeapi.connect_to('Leaf3')
```

Výpis 4.10: Příklad příkazů připojení do směrovačů

Tab. 4.7: Postup a moduly API

Postup	Co je potřeba udělat	Modul
1	Ethernet1-3 vypnout funkce swichport	api('switchports')
2	Konfigurace IP adres	api('ipinterfaces')
3	Zapnuti ip routing	execute(['CLI příkaz'])
4	Vytvoření a konfigurace BGP	api('bgp')
5	Vytvoření Vlan	api('vlans')
6	Konfigurace IP adres Vlan	api('ipinterfaces')
7	Přidání Vlan do Ethernet	api('switchports')
8	Vytvoření a konfigurace IP adres Loopback	api('ipinterfaces')
9	Vytvoření VXLAN	api('ipinterfaces')
10	Konfigurace VXLAN	api('vlans')

Tab. 4.8: Možnosti modulu api('vlans')

Modul	Volání	Co dělá
api('vlans')	create(vid)	Vytvoří nový VLAN

Tab. 4.9: Možnosti modulu api('switchports')

Modul	Volání	Co dělá
api('switchports')	delete(name)	Odstranit logický switchport rozhraní
	create(name)	Vytvoří nové rozhraní logické vrstvy 2
	set_mode(Eth1, value=access)	Konfiguruje režim switchport
	set_access_vlan(Eth1, value=vlan_id)	Konfiguruje přístupový vlan pro switchport

Tab. 4.10: Možnosti modulu api('ipinterfaces')

Modul	Volání	Co dělá
api('ipinterfaces')	set_address(name, value=None)	Konfiguruje IP adresu rozhraní
	create(name)	Vytvoří nové IP rozhraní

Tab. 4.11: Možnosti modulu execute'CLI příkaz'

Modul	Volání	Co dělá
execute(['CLI příkaz'])	execute(['enable', 'configure' , 'ip routing'])	Posíla CLI příkazy

Tab. 4.12: Možnosti modulu api('bgp')

Modul	Volání	Co dělá
api('bgp')	create(bgp_as)	Vytvoření AS
	configure_bgp	Posíla CLI příkazy
	set_maximum_paths(max_path=2, max_ecmp_path=2,	Configurace ECMP
api ('bgp.neighbors)	create(ip_adsress)	Vytvoření IP adresy souseda
	set_remote_as(ip_adsress,value='as')	Přiřadit sousedu AS
	set_shutdown(ip_adsress, disable=True)	Zapnutí sousedství

Tab. 4.13: Možnosti modulu api('interfaces')

Modul	Volání	Co dělá
api('interfaces')	add_vtep(Vxlan1, ip adres)	Přidá nový koncový bod VTEP
	set_source_interface(Vxlan1, value=Lo1)	Konfiguruje zdrojové rozhraní VXLAN
	update_vlan(vxlan1,VLAN, vni)	Přidá nový vlan do mapování VNI

```
1 # start by importing the library
2 import pyeapi
3 #promeny
4 eth1 = 'Ethernet1'
5 #Pripojeni
6 VEOSCL = pyeapi.connect(host='172.25.35.5', transport='https', username='lab',
      password='lab')
7 #Uvidet vesion EOS
8 VEOSCL.execute(['enable', 'show version'])
9
```

Výpis 4.11: Příklad skriptu konfigurace přes CLI

Tab. 4.14: Data pro konfiguraci rozhraní směrovačů a BGP

Směrovač	BGP AS	Rozhraní	IP adres /24	IP adres sousedů a AS
Spine1	150	Ethernet1	10.1.1.1	10.1.1.2 210
		Ethernet2	10.1.2.1	10.1.2.2 220
		Ethernet3	10.1.3.1	10.1.3.2 230
Spine2		Ethernet1	20.1.1.1	20.1.1.2 210
		Ethernet2	20.1.2.1	20.1.2.2 220
		Ethernet3	20.1.3.1	20.1.3.2 230
Leaf1	210	Ethernet1	10.1.1.2	10.1.1.1 150
		Ethernet2	10.1.1.2	
		Vlan 30	192.168.30.254	20.1.1.1 150
		Vlan 40	192.168.40.254	
		Loopback1	110.110.110.1	
Leaf2	220	Ethernet1	10.1.2.2	10.1.2.1 150
		Ethernet2	20.1.2.2	
		Vlan 30	192.168.30.254	20.1.2.1 150
		Vlan 40	192.168.40.254	
		Loopback1	120.120.120.1	
Leaf3	230	Ethernet1	10.1.3.2	10.1.3.1 150
		Ethernet2	20.1.3.2	
		Vlan 50	192.168.50.254	20.1.3.1 150
		Loopback1	130.130.130.1	

```
1  # start by importing the library
2  import pyeapi
3  #promeny
4  eth1 = 'Ethernet1'
5  #Pripojeni
6  pyeapi.load_config('location.conf')
7  VEOS = pyeapi.connect_to('AristaVEOS1')
8  #Konfigurace ip adresy Ethernet1
9  print('Set ip addres for Ethernet1:')
10 ipinterface = VEOS.api('ipinterfaces')
11 ipinterface.set_address(eth1, value='10.10.20.1/24')
12 prt2 = ipinterface.get(eth1)
13 print(prt2['name'], prt2['address'])
14
```

Výpis 4.12: Příklad skriptu konfigurace IP adresy

Tab. 4.15: Data pro konfiguraci Vlan a VXLAN

Směrovač	Rozhraní	IP adres /24	Vlan	Vtep /24	Vlan do VNI
Leaf1	Ethernet3	-	30	-	-
	Ethernet4	-	40	-	-
	Vlan 30	192.168.30.254	-	-	-
	Vlan 40	192.168.40.254	-	-	-
	Vxlan1	-	-	Loopback1 120.120.120.1, 130.130.130.1	30 do 300 40 do 400
Leaf2	Ethernet3	-	30	-	-
	Ethernet4	-	40	-	-
	Vlan 30	192.168.30.254	-	-	-
	Vlan 40	192.168.40.254	-	-	-
	Vxlan1	-	-	Loopback1 110.110.110.1, 130.130.130.1	30 do 300 40 do 400
Leaf3	Ethernet5	-	50	-	-
	Vlan 50	192.168.50.254		-	-

(d) Pro ověření toho, jak proběhlo konfigurování lze využít SSH připojení do směrovače, uživatelský účet a heslo je **Lab** na všech směrovacích. Po připojení je nezbytné využít příkaz **enable** a pak **show running-config** a hned se zobrazí aktuální konfigurace směrovače. Příkazem **show ip routin** se dá uvidět směrovací tabulku, **show ip interface brief** – IP adresy aktivních rozhraní, **show ip bgb neibors** – vidět IP adresy sousedů, které využívají BGP. Pokud před restartováním směrovače není využit příkaz **write**, konfigurace nebude uložena, po zapnutí se rozběhne poslední zapsaná konfigurace.

(e) Pokud je všechno nakonfigurováno, je potřeba ověřit komunikace mezi VPC pomocí příkazu **ping**. Pokud v EVE-NG dvakrát zmáčknout na VPC proběhne automatické připojení k němu.

(f) Po dokončení skriptu je potřeba ho ukázat a popsat, co dělá.

2. Ansible

(a) Nyní lze přejít do nastavení směrovačů pomocí Ansible. Je potřeba provést obnovu konfigurace sítě v EVE-NG, nasledně opakovat krok **1a** a pak vytvořit skript, který nakonfiguruje síť „Leaf-Spine“ Layer 3/ECMP a VXLAN stejně, jako probíhalo konfigurování přes Python skript.

(b) Připojení do Ansible bude probíhat pomocí SSH klientu PuTTy, který je

na ploše. Ansible už je nainstalován na obrazu disku virtuálního serveru LAB2_Ansible. Spustit se dá pomocí Wmplayer. IP adres serveru je `172.25.35.170`, uživatelské jméno a heslo je **Lab**.

(c) Cesta k hlavnímu souboru Ansible je **/etc/ansible/**.Tam je potřeba vytvořit programovací skript a prostředí složek, ve kterých budou pomocná data. Jak má vypadat prostředí, je vidět na Obr. 4.12. Pomocí příkazu

```
Ansible/ ..... kořenový adresář Ansible
├── group_vars
│   ├── api.yml. ..... Konfigurace spojení přes api
│   └── cli.yml. ..... konfigurace spojení přes cli
├── templates ..... šablony
│   ├── l2vxlan.j2
│   ├── ip_routing.j2
│   └── maximum_paths.j2
├── host ..... IP adresy a hesla směrovačů
└── playbook.yml ..... konfigurační skript
```

Obr. 4.12: Prostředí složek Ansible.

nano, lze vytvořit konfigurační soubor. Koncovka souboru má být **.yml**, například **nano /etc/ansible/playbook.yml**. Pro zjednodušení, prostředí Ansible už je připravené a vypadá přesně tak, jak je zobrazeno na Obr. 4.12. Tělo skriptu **playbook.yml** je také připravené podle tabulky 4.16 a změnit ho může příkaz **nano**. V těle skriptu je potřeba měnit jenom položky, které jsou označeny **#pridat**. Ansible využívá YAML Syntax, je potřeba respektovat strukturu textu pro YAML. Příkazem **ansible-playbook LAB.yml** je možné spustit skript. Funguje pouze, pokud se nachází ve složce **/etc/ansible/**.Postup konfigurování je vysvětlen v tabulce 4.16. Data pro konfigurování byla uvedena v prvním úkolu, viz tabulky 4.14, 4.15. Ve výpisech 4.13 a 4.14 jsou příklady jednoduchých skriptů [18].

(d) Dále v tabulkách 4.17 a 4.18 budou popsané možnosti modulů, které je potřeba využít [21].

(e) Pro ověření konfigurování je potřeba využít SSH spojení se směrovačem, viz **1d**.

(f) Pokud všechno nakonfigurováno, je potřeba opakovat krok **1e**.

(g) Po dokončení skriptu je potřeba ho ukázat a popsat, co dělá.

Tab. 4.16: Postup a moduly Ansible

Postup	Co je potřeba udělat	Modul
1	Ethernet1-3 vypnout funkce swich-port	eos_l2_interface
2	Konfigurace IP adres	eos_l3_interface
3	Zapnutí ip routing	eos_config
4	Vytvoření a konfigurace BGP	eos_bgp
5	Vytvoření Vlan a přidání do Ether-net	eos_vlan
6	Konfigurace IP adres Vlan	eos_l3_interface
7	Vytvoření a konfigurace IP adres Loopback	eos_interface
8	Vytvoření VXLAN a konfigurace	eos_config

Tab. 4.17: Možnosti modulu eos_l2 a l3_interface, eos_vlan, eos_config

Modul	Volání	Co dělá
eos_l2_interface	name: state:	Odstranit logický switchport rozhraní
eos_l3_interface	name: ipv4:	Nastavení IP adresy pro rozhraní
eos_vlan	vlan_id: state: present interfaces:	Konfiguruje přístupový vlan pro Ethernet
eos_config	eos_config:	Posílá CLI příkazy

Tab. 4.18: Možnosti modulu eos_bgp

Modul		Volání	Co dělá	
eos_bgp	config	bgp_as:	Kofiguruje AS	-
		- maximum_ prefix:	Konfiguruje ECMP	-
		neighbors	- neighbor: remote_as:	Konfiguruje IP adres souseda a AS
		redistribute:	- protocol: connected	Určuje protokol pro konfiguraci informací o redistribuci.

```
---
- hosts: spine2
  gather_facts: no
  tasks:
   - name: Set ethernet1 IPv4 address  ETh 1-3
     eos_l3_interface:
      name: "{{ item.name }}"
      ipv4: "{{ item.ip }}"
     loop:
        - { name: 'Ethernet1', ip: '20.1.1.1/24' }
        - { name: 'Ethernet2', ip: '20.1.2.1/24' }
        - { name: 'Ethernet3', ip: '20.1.3.1/24' }
```

Výpis 4.13: Příklad skriptu pro Ansible modul eos_l3_interface

```
---
- hosts: leaf1
  connection: local
  gather_facts: no

  tasks:
  - name: Configure L2 VXLAN
    eos_config:
      src: iprouting.j2
###########END_Script#########################
###########Pro iprouting.j2, je potreba####
######vytvorit template kam dat iprouting.j2#
ip routing
##############################################
```

Výpis 4.14: Příklad skriptu pro Ansible modul eos_config

4.3.7 Kontrolní otázky

- Na které vrstvě existuje v úloze VXLAN?
- Jaký protokol byl využit pro předání směrovacích informací mezi směrovači?
- V jaké úloze byl vyvolán modul pomocí SSH?

4.3.8 Úklid pracoviště

- Smazat skripty, které byly během úlohy vytvořeny.
- Vypnout virtuální stanice v Wmplayeru.

4.4 Laboratorní úloha 2 – pokyny pro vyučující

Skript pro Python

```
####################Telo_skriptu################################################
# start by importing the library
import pyeapi
#promeny
eth1 = 'Ethernet1'
eth2 = 'Ethernet2'
eth3 = 'Ethernet3'
eth4 = 'Ethernet4'
eth5 = 'Ethernet5'
# import host ip adres, login, password
pyeapi.load_config('host.conf')
# create a node object by specifying the node to work with
Spine1 = pyeapi.connect_to('Spine1')
Spine2 = pyeapi.connect_to('Spine2')
Leaf1 = pyeapi.connect_to('Leaf1')
Leaf2 = pyeapi.connect_to('Leaf2')
Leaf3 = pyeapi.connect_to('Leaf3')
####################Postup_cislo_1_############################################
#-----Vypnuti_switchport_pro_Ethernet1-3_na_smerovaci_Spine_1_------
swmode = Spine1.api('switchports')
prt1 = swmode.getall()
print('--------Spine1--------')
print('-Spine1-----Rozhrabni co jsou v switchports:', list(prt1.keys()))
print('-Spine1-Odstraneni ze switchports:', eth1, ',', eth2,',', eth3,'.')
swmode.delete(eth1)
swmode.delete(eth2)
swmode.delete(eth3)
prt1 = swmode.getall()
print('-Spine1--Po odstraneni co zbyva v switchports:', list(prt1.keys()))
#-----Vypnuti_switchport_pro_Ethernet1-3_na_smerovaci_Spine_2_------
swmode = Spine2.api('switchports')
prt1 = swmode.getall()
print('--------Spine2--------')
print('-Spine2-----Rozhrabni co jsou v switchports:', list(prt1.keys()))
print('-Spine2-Odstraneni ze switchports:', eth1, ',', eth2,',', eth3,'.')
swmode.delete(eth1)
swmode.delete(eth2)
swmode.delete(eth3)
prt1 = swmode.getall()
print('-Spine2--Po odstraneni co zbyva v switchports:', list(prt1.keys()))
#-----Vypnuti_switchport_pro_Ethernet1-2_na_smerovaci_Leaf_1_------
swmode = Leaf1.api('switchports')
prt1 = swmode.getall()
print('--------Leaf1--------')
print('-Leaf1-----Rozhrabni co jsou v switchports:', list(prt1.keys()))
print('-Leaf1-Odstraneni ze switchports:', eth1, ',', eth2,'.')
swmode.delete(eth1)
swmode.delete(eth2)
prt1 = swmode.getall()
print('-Leaf1--Po odstraneni co zbyva v switchports:', list(prt1.keys()))
#-----Vypnuti_switchport_pro_Ethernet1-2_na_smerovaci_Leaf_2_------
swmode = Leaf2.api('switchports')
prt1 = swmode.getall()
print('--------Leaf2--------')
print('-Leaf2-----Rozhrabni co jsou v switchports:', list(prt1.keys()))
```

```
print('-Leaf2-Odstraneni ze switchports:', eth1, ',', eth2,'.')
swmode.delete(eth1)
swmode.delete(eth2)
prt1 = swmode.getall()
print('-Leaf2--Po odstraneni co zbyva v switchports:', list(prt1.keys()))
#-----Vypnuti_switchport_pro_Ethernet1-2_na_smerovaci_Leaf_3_------
swmode = Leaf3.api('switchports')
prt1 = swmode.getall()
print('--------Leaf3--------')
print('-Leaf3-----Rozhrabni co jsou v switchports:', list(prt1.keys()))
print('-Leaf3-Odstraneni ze switchports:', eth1, ',', eth2,'.')
swmode.delete(eth1)
swmode.delete(eth2)
prt1 = swmode.getall()
print('-Leaf3--Po odstraneni co zbyva v switchports:', list(prt1.keys()))
####################Postup_cislo_1_KONEC_###################################
####################Postup_cislo_2_#########################################
#-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Spine_1_------
print('--------Spine1--------')
print('Konfigurovani IP adres:')
ipinterface = Spine1.api('ipinterfaces')
ipinterface.set_address(eth1, value='10.1.1.1/24')
ipinterface.set_address(eth2, value='10.1.2.1/24')
ipinterface.set_address(eth3, value='10.1.3.1/24')
prt2 = ipinterface.get(eth1)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth2)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth3)
print(prt2['name'], ':', prt2['address'])
#-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Spine_2_------
print('--------Spine2--------')
print('Konfigurovani IP adres:')
ipinterface = Spine2.api('ipinterfaces')
ipinterface.set_address(eth1, value='20.1.1.1/24')
ipinterface.set_address(eth2, value='20.1.2.1/24')
ipinterface.set_address(eth3, value='20.1.3.1/24')
prt2 = ipinterface.get(eth1)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth2)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth3)
print(prt2['name'], ':', prt2['address'])
#-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
print('Konfigurovani IP adres:')
ipinterface = Leaf1.api('ipinterfaces')
ipinterface.set_address(eth1, value='10.1.1.2/24')
ipinterface.set_address(eth2, value='20.1.1.2/24')
prt2 = ipinterface.get(eth1)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth2)
print(prt2['name'], ':', prt2['address'])
#-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Leaf_2_------
print('--------Leaf2--------')
print('Konfigurovani IP adres:')
ipinterface = Leaf2.api('ipinterfaces')
ipinterface.set_address(eth1, value='10.1.2.2/24')
ipinterface.set_address(eth2, value='20.1.2.2/24')
```

```
prt2 = ipinterface.get(eth1)
prt2 = ipinterface.get(eth1)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth2)
print(prt2['name'], ':', prt2['address'])
#-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Leaf_3_------
print('--------Leaf3--------')
print('Konfigurovani IP adres:')
ipinterface = Leaf3.api('ipinterfaces')
ipinterface.set_address(eth1, value='10.1.3.2/24')
ipinterface.set_address(eth2, value='20.1.3.2/24')
prt2 = ipinterface.get(eth1)
print(prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(eth2)
print(prt2['name'], ':', prt2['address'])
####################Postup_cislo_2_KONEC_###################################
####################Postup_cislo_3_#########################################
#-----konfigurovani_ip_routing_na_smerovaci_Spine_1_------
print('--------Spine1--------')
print('Konfigurovani ip_rounting')
Spine1.run_commands(['enable', 'configure', 'ip routing'])
#-----konfigurovani_ip_routing_na_smerovaci_Spine_2_------
print('--------Spine2--------')
print('Konfigurovani ip_rounting')
Spine2.run_commands(['enable', 'configure', 'ip routing'])
#-----konfigurovani_ip_routing_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
print('Konfigurovani ip_rounting')
Leaf1.run_commands(['enable', 'configure', 'ip routing'])
#-----konfigurovani_ip_routing_na_smerovaci_Leaf_2_------
print('--------Leaf2--------')
print('Konfigurovani ip_rounting')
Leaf2.run_commands(['enable', 'configure', 'ip routing'])
#-----konfigurovani_ip_routing_na_smerovaci_Leaf_3_------
print('--------Leaf3--------')
print('Konfigurovani ip_rounting')
Leaf3.run_commands(['enable', 'configure', 'ip routing'])
####################Postup_cislo_3_KONEC_###################################
####################Postup_cislo_4_#########################################
#-----konfigurovani_BGP_na_smerovaci_Spine_1_------
print('--------Spine1--------')
bgpc = Spine1.api('bgp')
AS = '150'
soudes210_ip_adres = '10.1.1.2'
soudes220_ip_adres = '10.1.2.2'
soudes230_ip_adres = '10.1.3.2'
soused_Leaf1_AS = '210'
soused_Leaf2_AS = '220'
soused_Leaf3_AS = '230'
bgpc.create(AS)
bgpc.configure_bgp('redistribute connected')
bgpc.set_maximum_paths(max_path=2, max_ecmp_path=2)
bgpc.neighbors.create(soudes210_ip_adres)
bgpc.neighbors.set_remote_as(soudes210_ip_adres,value=soused_Leaf1_AS)
bgpc.neighbors.set_shutdown(soudes210_ip_adres, disable=True)
bgpc.neighbors.create(soudes220_ip_adres)
bgpc.neighbors.set_remote_as(soudes220_ip_adres,value=soused_Leaf2_AS)
bgpc.neighbors.set_shutdown(soudes220_ip_adres, disable=True)
bgpc.neighbors.create(soudes230_ip_adres)
```

```
bgpc.neighbors.set_remote_as(soudes230_ip_adres,value=soused_Leaf3_AS)
bgpc.neighbors.set_shutdown(soudes230_ip_adres, disable=True)
bgpsoused = bgpc.neighbors.get(soudes210_ip_adres)
bgpsoused1 = bgpc.neighbors.get(soudes220_ip_adres)
bgpsoused2 = bgpc.neighbors.get(soudes230_ip_adres)
bgpAS = bgpc.get()
print('-Je nakonfigurovano BGP, kde AS =', bgpAS['bgp_as'])
print('--Soused:', bgpsoused['name'],' AS =', bgpsoused['remote_as'])
print('--Soused:', bgpsoused1['name'],' AS =', bgpsoused1['remote_as'])
print('--Soused:', bgpsoused2['name'],' AS =', bgpsoused2['remote_as'])
print('--ECMP se rovna:', bgpAS['maximum_paths'])
#-----konfigurovani_BGP_na_smerovaci_Spine_2_------
print('--------Spine2--------')
bgpc = Spine2.api('bgp')
AS = '150'
soudes210_ip_adres = '20.1.1.2'
soudes220_ip_adres = '20.1.2.2'
soudes230_ip_adres = '20.1.3.2'
soused_Leaf1_AS = '210'
soused_Leaf2_AS = '220'
soused_Leaf3_AS = '230'
bgpc.create(AS)
bgpc.configure_bgp('redistribute connected')
bgpc.set_maximum_paths(max_path=2, max_ecmp_path=2)
bgpc.neighbors.create(soudes210_ip_adres)
bgpc.neighbors.set_remote_as(soudes210_ip_adres,value=soused_Leaf1_AS)
bgpc.neighbors.set_shutdown(soudes210_ip_adres, disable=True)
bgpc.neighbors.create(soudes220_ip_adres)
bgpc.neighbors.set_remote_as(soudes220_ip_adres,value=soused_Leaf2_AS)
bgpc.neighbors.set_shutdown(soudes220_ip_adres, disable=True)
bgpc.neighbors.create(soudes230_ip_adres)
bgpc.neighbors.set_remote_as(soudes230_ip_adres,value=soused_Leaf3_AS)
bgpc.neighbors.set_shutdown(soudes230_ip_adres, disable=True)
bgpsoused = bgpc.neighbors.get(soudes210_ip_adres)
bgpsoused1 = bgpc.neighbors.get(soudes220_ip_adres)
bgpsoused2 = bgpc.neighbors.get(soudes230_ip_adres)
bgpAS = bgpc.get()
print('-Je nakonfigurovano BGP, kde AS =', bgpAS['bgp_as'])
print('--Soused:', bgpsoused['name'],' AS =', bgpsoused['remote_as'])
print('--Soused:', bgpsoused1['name'],' AS =', bgpsoused1['remote_as'])
print('--Soused:', bgpsoused2['name'],' AS =', bgpsoused2['remote_as'])
print('--ECMP se rovna:', bgpAS['maximum_paths'])
#-----konfigurovani_BGP_na_smerovaci_Leaf1_------
print('--------Leaf1--------')
bgpc = Leaf1.api('bgp')
AS = '210'
soudes_Spine1_ip_adres = '10.1.1.1'
soudes_Spine2_ip_adres = '20.1.1.1'
soused_Spine1_AS = '150'
soused_Spine2_AS = '150'
bgpc.create(AS)
bgpc.configure_bgp('redistribute connected')
bgpc.set_maximum_paths(max_path=2, max_ecmp_path=2)
bgpc.neighbors.create(soudes_Spine1_ip_adres)
bgpc.neighbors.set_remote_as(soudes_Spine1_ip_adres,value=soused_Spine1_AS)
bgpc.neighbors.set_shutdown(soudes_Spine1_ip_adres, disable=True)
bgpc.neighbors.create(soudes_Spine2_ip_adres)
bgpc.neighbors.set_remote_as(soudes_Spine2_ip_adres,value=soused_Spine2_AS)
bgpc.neighbors.set_shutdown(soudes_Spine2_ip_adres, disable=True)
```

```
bgpsoused = bgpc.neighbors.get(soudes_Spine1_ip_adres)
bgpsoused1 = bgpc.neighbors.get(soudes_Spine2_ip_adres)
bgpAS = bgpc.get()
print('-Je nakonfigurovano BGP, kde AS =', bgpAS['bgp_as'])
print('--Soused:', bgpsoused['name'],' AS =', bgpsoused['remote_as'])
print('--Soused:', bgpsoused1['name'],' AS =', bgpsoused1['remote_as'])
print('--ECMP se rovna:', bgpAS['maximum_paths'])
#-----konfigurovani_BGP_na_smerovaci_Leaf2_------
print('--------Leaf2--------')
bgpc = Leaf2.api('bgp')
AS = '220'
soudes_Spine1_ip_adres = '10.1.2.1'
soudes_Spine2_ip_adres = '20.1.2.1'
soused_Spine1_AS = '150'
soused_Spine2_AS = '150'
bgpc.create(AS)
bgpc.configure_bgp('redistribute connected')
bgpc.set_maximum_paths(max_path=2, max_ecmp_path=2)
bgpc.neighbors.create(soudes_Spine1_ip_adres)
bgpc.neighbors.set_remote_as(soudes_Spine1_ip_adres,value=soused_Spine1_AS)
bgpc.neighbors.set_shutdown(soudes_Spine1_ip_adres, disable=True)
bgpc.neighbors.create(soudes_Spine2_ip_adres)
bgpc.neighbors.set_remote_as(soudes_Spine2_ip_adres,value=soused_Spine2_AS)
bgpc.neighbors.set_shutdown(soudes_Spine2_ip_adres, disable=True)
bgpsoused = bgpc.neighbors.get(soudes_Spine1_ip_adres)
bgpsoused1 = bgpc.neighbors.get(soudes_Spine2_ip_adres)
bgpAS = bgpc.get()
print('-Je nakonfigurovano BGP, kde AS =', bgpAS['bgp_as'])
print('--Soused:', bgpsoused['name'],' AS =', bgpsoused['remote_as'])
print('--Soused:', bgpsoused1['name'],' AS =', bgpsoused1['remote_as'])
print('--ECMP se rovna:', bgpAS['maximum_paths'])
#-----konfigurovani_BGP_na_smerovaci_Leaf3_------
print('--------Leaf3--------')
bgpc = Leaf3.api('bgp')
AS = '230'
soudes_Spine1_ip_adres = '10.1.3.1'
soudes_Spine2_ip_adres = '20.1.3.1'
soused_Spine1_AS = '150'
soused_Spine2_AS = '150'
bgpc.create(AS)
bgpc.configure_bgp('redistribute connected')
bgpc.set_maximum_paths(max_path=2, max_ecmp_path=2)
bgpc.neighbors.create(soudes_Spine1_ip_adres)
bgpc.neighbors.set_remote_as(soudes_Spine1_ip_adres,value=soused_Spine1_AS)
bgpc.neighbors.set_shutdown(soudes_Spine1_ip_adres, disable=True)
bgpc.neighbors.create(soudes_Spine2_ip_adres)
bgpc.neighbors.set_remote_as(soudes_Spine2_ip_adres,value=soused_Spine2_AS)
bgpc.neighbors.set_shutdown(soudes_Spine2_ip_adres, disable=True)
bgpsoused = bgpc.neighbors.get(soudes_Spine1_ip_adres)
bgpsoused1 = bgpc.neighbors.get(soudes_Spine2_ip_adres)
bgpAS = bgpc.get()
print('-Je nakonfigurovano BGP, kde AS =', bgpAS['bgp_as'])
print('--Soused:', bgpsoused['name'],' AS =', bgpsoused['remote_as'])
print('--Soused:', bgpsoused1['name'],' AS =', bgpsoused1['remote_as'])
print('--ECMP se rovna:', bgpAS['maximum_paths'])
####################Postup_cislo_4_KONEC_#####################################
####################Postup_cislo_5_###########################################
#-----konfigurovani_Vlan_30_40_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
```

```
vlan = Leaf1.api('vlans')
vlan_id = '30'
vlan_id1 = '40'
vlan.create(vlan_id)
vlan.create(vlan_id1)
typepr = vlan.getall()
print('Vlans co jsou na smerovacu:', list(typepr.keys()))
#-----konfigurovani_Vlan_30_40_na_smerovaci_Leaf_2_------
print('--------Leaf2--------')
vlan = Leaf2.api('vlans')
vlan_id = '30'
vlan_id1 = '40'
vlan.create(vlan_id)
vlan.create(vlan_id1)
typepr = vlan.getall()
print('Vlans co jsou na smerovacu:', list(typepr.keys()))
#-----konfigurovani_Vlan_50_na_smerovaci_Leaf_3_------
print('--------Leaf3--------')
vlan = Leaf3.api('vlans')
vlan_id = '50'
vlan.create(vlan_id)
typepr = vlan.getall()
print('Vlans co jsou na smerovacu:', list(typepr.keys()))
####################Postup_cislo_5_KONEC_###################################
####################Postup_cislo_6_#########################################
#-----konfigurovani_IP_adres_Vlans_30_40_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
ipinterface = Leaf1.api('ipinterfaces')
vlan_name = 'Vlan30'
vlan_name1 = 'Vlan40'
ip_adres_for_vlan_id = '192.168.30.254/24'
ip_adres_for_vlan_id1 = '192.168.40.254/24'
ipinterface.set_address(vlan_name, value = ip_adres_for_vlan_id)
ipinterface.set_address(vlan_name1, value = ip_adres_for_vlan_id1)
prt2 = ipinterface.get(vlan_name)
print('Nakonfigurovani Vlans:')
print('Nazev:',prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(vlan_name1)
print('Nazev:',prt2['name'], ':', prt2['address'])
#-----konfigurovani_IP_adres_Vlans_30_40_na_smerovaci_Leaf_2_------
print('--------Leaf1--------')
ipinterface = Leaf2.api('ipinterfaces')
vlan_name = 'Vlan30'
vlan_name1 = 'Vlan40'
ip_adres_for_vlan_id = '192.168.30.254/24'
ip_adres_for_vlan_id1 = '192.168.40.254/24'
ipinterface.set_address(vlan_name, value = ip_adres_for_vlan_id)
ipinterface.set_address(vlan_name1, value = ip_adres_for_vlan_id1)
prt2 = ipinterface.get(vlan_name)
print('Nakonfigurovani Vlans:')
print('Nazev:',prt2['name'], ':', prt2['address'])
prt2 = ipinterface.get(vlan_name1)
print('Nazev:',prt2['name'], ':', prt2['address'])
#-----konfigurovani_IP_adres_Vlans_30_40_na_smerovaci_Leaf_3_------
print('--------Leaf3--------')
ipinterface = Leaf3.api('ipinterfaces')
vlan_name = 'Vlan50'
ip_adres_for_vlan_id = '192.168.50.254/24'
ipinterface.set_address(vlan_name, value = ip_adres_for_vlan_id)
```

```
prt2 = ipinterface.get(vlan_name)
print('Nakonfigurovani Vlans:')
print('Nazev:',prt2['name'], ':', prt2['address'])
####################Postup_cislo_6_KONEC_#####################################
####################Postup_cislo_7_###########################################
#-----Pridani_Vlan_30_40_do_ethernet3-4_Leaf1------
swmode = Leaf1.api('switchports')
print('--------Leaf1--------')
vlan30_id = '30'
vlan40_id = '40'
swmode.set_mode(eth3, value='access')
swmode.set_access_vlan(eth3, value = vlan30_id)
swmode.set_mode(eth4, value='access')
swmode.set_access_vlan(eth4, value = vlan40_id)
prt2 = swmode.get(eth3)
print(prt2['name'], ':', prt2['mode'], ' Vlan', prt2['access_vlan'])
prt2 = swmode.get(eth4)
print(prt2['name'], ':', prt2['mode'], ' Vlan', prt2['access_vlan'])
#-----Pridani_Vlan_30_40_do_ethernet3-4_Leaf2------
swmode = Leaf2.api('switchports')
print('--------Leaf2--------')
vlan30_id = '30'
vlan40_id = '40'
swmode.set_mode(eth3, value='access')
swmode.set_access_vlan(eth3, value = vlan30_id)
swmode.set_mode(eth4, value='access')
swmode.set_access_vlan(eth4, value = vlan40_id)
prt2 = swmode.get(eth3)
print(prt2['name'], ':', prt2['mode'], ' Vlan', prt2['access_vlan'])
prt2 = swmode.get(eth4)
print(prt2['name'], ':', prt2['mode'], ' Vlan', prt2['access_vlan'])
#-----Pridani_Vlan_50_do_ethernet5_Leaf3------
swmode = Leaf3.api('switchports')
print('--------Leaf3--------')
vlan50_id = '50'
swmode.set_mode(eth5, value='access')
swmode.set_access_vlan(eth5, value = vlan50_id)
prt2 = swmode.get(eth5)
print(prt2['name'], ':', prt2['mode'], ' Vlan', prt2['access_vlan'])
####################Postup_cislo_7_KONEC_#####################################
####################Postup_cislo_8_###########################################
#-----konfigurovani_IP_adres_loopback1_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
ipinterface = Leaf1.api('ipinterfaces')
loopback_name = 'Loopback1'
ip_adres_loopback1 = '110.110.110.1/24'
ipinterface.create(loopback_name)
ipinterface.set_address(loopback_name, value = ip_adres_loopback1)
prt2 = ipinterface.get(loopback_name)
print('Nakonfigurovan Loopback1:')
print('Nazev:',prt2['name'], ':', prt2['address'])
#-----konfigurovani_IP_adres_loopback1_na_smerovaci_Leaf_2_------
print('--------Leaf2--------')
ipinterface = Leaf2.api('ipinterfaces')
loopback_name = 'Loopback1'
ip_adres_loopback1 = '120.120.120.1/24'
ipinterface.create(loopback_name)
ipinterface.set_address(loopback_name, value = ip_adres_loopback1)
prt2 = ipinterface.get(loopback_name)
```

```
print('Nakonfigurovan Loopback1:')
print('Nazev:',prt2['name'], ':', prt2['address'])
####################Postup_cislo_8_KONEC_###################################
####################Postup_cislo_9_#########################################
#-----Vytvoreni_VXLAN1_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
ipinterface = Leaf1.api('ipinterfaces')
VXLAN_name = 'VXLAN1'
ipinterface.create(VXLAN_name)
print('Vytvoreni VXLAN1')
#-----Vytvoreni_VXLAN1_na_smerovaci_Leaf_2_------
print('--------Leaf2--------')
ipinterface = Leaf2.api('ipinterfaces')
VXLAN_name = 'VXLAN1'
ipinterface.create(VXLAN_name)
print('Vytvoreni VXLAN1')
####################Postup_cislo_9_KONEC_###################################
####################Postup_cislo_10_########################################
#-----konfigurovani_VXLAN1_na_smerovaci_Leaf_1_------
print('--------Leaf1--------')
vxlan = Leaf1.api('interfaces')
VXLAN_name = 'VXLAN1'
Loopback1_name = 'Loopback1'
vlan30_id = 30
vlan40_id = 40
vtep_ip_adres = '120.120.120.1'
vxlan.set_source_interface(VXLAN_name, value = Loopback1_name)
vxlan.update_vlan(VXLAN_name, vlan30_id, vlan30_id*10)
vxlan.update_vlan(VXLAN_name, vlan40_id, vlan40_id*10)
vxlan.add_vtep(VXLAN_name, vtep_ip_adres)
print('Nakonfigurovan VXLAN soused vni a vtep')
prt2_v = vxlan.get(VXLAN_name)
print('Nazev:',prt2_v['name'], ',Vypnuty:', prt2_v['source_interface'])
#-----konfigurovani_VXLAN1_na_smerovaci_Leaf_2_------
print('--------Leaf2--------')
vxlan = Leaf2.api('interfaces')
VXLAN_name = 'VXLAN1'
Loopback1_name = 'Loopback1'
vlan30_id = 30
vlan40_id = 40
vtep_ip_adres = '110.110.110.1'
vxlan.set_source_interface(VXLAN_name, value = Loopback1_name)
vxlan.update_vlan(VXLAN_name, vlan30_id, vlan30_id*10)
vxlan.update_vlan(VXLAN_name, vlan40_id, vlan40_id*10)
vxlan.add_vtep(VXLAN_name, vtep_ip_adres)
print('Nakonfigurovan VXLAN soused vni a vtep')
prt2_v = vxlan.get(VXLAN_name)
print('Nazev:',prt2_v['name'], ',Vypnuty:', prt2_v['source_interface'])
####################Postup_cislo_10_KONEC_##################################
############################################################################
```

Výpis 4.15: Příklad skriptu v jazyce Python pro laboratorní úlohu číslo 2.

Skript pro Ansible

Kvůli tomu, že modul eos_bgp hlásí chybu ve chvíli, kdy funguje přes eAPI, bylo využito SSH připojení. Ani modul eos_bgp neobsahuje možnosti, jak nakonfigurovat ECMP, proto bylo využito eos_config a k tomu šablona maximum_paths.j2.

Následně byl eos_config využit ještě se šablonami l2vxlan.j2 a ip_routing. Šablona l2vxlan.j2 obsahuje konfigurace VXLAN a ip_routing jen jeden přikaz zapnutí routování.

```
---
####################Telo_skriptu##############################################
####################Postup_cislo_1_###########################################
#-----Vypnuti_switchport_pro_Ethernet1-3_na_smerovaci_Spine_1_------
- hosts: spine1
  gather_facts: no
  tasks:
   - name: Spine1 off switchport Eth 1-3
     eos_l2_interface:
       name: "{{ item }}"
       state: absent
     loop:
            - Ethernet1
            - Ethernet2
            - Ethernet3
#-----Vypnuti_switchport_pro_Ethernet1-3_na_smerovaci_Spine_2_------
- hosts: spine2
  gather_facts: no
  tasks:
   - name: Spine2 off switchport Eth 1-3
     eos_l2_interface:
       name: "{{ item }}"
       state: absent
     loop:
            - Ethernet1
            - Ethernet2
            - Ethernet3
#-----Vypnuti_switchport_pro_Ethernet1-2_na_smerovaci_Leaf_1_------
- hosts: leaf1
  gather_facts: no
  tasks:
   - name: Leaf1 off switchport Eth 1-2
     eos_l2_interface:
       name: "{{ item }}"
       state: absent
     loop:
            - Ethernet1
            - Ethernet2
#-----Vypnuti_switchport_pro_Ethernet1-2_na_smerovaci_Leaf_2_------
- hosts: leaf2
  gather_facts: no
  tasks:
   - name: Leaf2 off switchport Eth 1-2
     eos_l2_interface:
       name: "{{ item }}"
       state: absent
     loop:
            - Ethernet1
            - Ethernet2
#-----Vypnuti_switchport_pro_Ethernet1-2_na_smerovaci_Leaf_3_------
- hosts: leaf3
  gather_facts: no
  tasks:
```

```
54    - name: Leaf3 off switchport Eth 1-2
55      eos_l2_interface:
56        name: "{{ item }}"
57        state: absent
58      loop:
59            - Ethernet1
60            - Ethernet2
61 ####################Postup_cislo_1_KONEC_#################################
62 ####################Postup_cislo_2_#######################################
63 #-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Spine_1_------
64 - hosts: spine1
65   gather_facts: no
66   tasks:
67    - name: Spine1 konfigurovani IPv4 address  ETh 1-3
68      eos_l3_interface:
69       name: "{{ item.name }}"
70       ipv4: "{{ item.ip }}"
71      loop:
72         - { name: 'Ethernet1', ip: '10.1.1.1/24' }
73         - { name: 'Ethernet2', ip: '10.1.2.1/24' }
74         - { name: 'Ethernet3', ip: '10.1.3.1/24' }
75 #-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Spine_1_------
76 - hosts: spine2
77   gather_facts: no
78   tasks:
79    - name: Spine2 konfigurovani IPv4 address  ETh 1-3
80      eos_l3_interface:
81       name: "{{ item.name }}"
82       ipv4: "{{ item.ip }}"
83      loop:
84         - { name: 'Ethernet1', ip: '20.1.1.1/24' }
85         - { name: 'Ethernet2', ip: '20.1.2.1/24' }
86         - { name: 'Ethernet3', ip: '20.1.3.1/24' }
87 #-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Leaf_1_------
88 - hosts: leaf1
89   gather_facts: no
90   tasks:
91    - name: Leaf1 konfigurovani IPv4 address  ETh 1-2
92      eos_l3_interface:
93       name: "{{ item.name }}"
94       ipv4: "{{ item.ip }}"
95      loop:
96         - { name: 'Ethernet1', ip: '10.1.1.2/24' }
97         - { name: 'Ethernet2', ip: '20.1.1.2/24' }
98 #-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Leaf_2_------
99 - hosts: leaf2
100  gather_facts: no
101  tasks:
102   - name: Leaf2 konfigurovani IPv4 address  ETh 1-2
103     eos_l3_interface:
104      name: "{{ item.name }}"
105      ipv4: "{{ item.ip }}"
106     loop:
107        - { name: 'Ethernet1', ip: '10.1.2.2/24' }
108        - { name: 'Ethernet2', ip: '20.1.2.2/24' }
109 #-----konfigurovani_ip_adres_pro_Ethernet1-3_na_smerovaci_Leaf_3_------
110 - hosts: leaf3
111   gather_facts: no
112   tasks:
```

```
113     - name: Leaf3 konfigurovani IPv4 address  ETh 1-2
114       eos_l3_interface:
115        name: "{{ item.name }}"
116        ipv4: "{{ item.ip }}"
117       loop:
118          - { name: 'Ethernet1', ip: '10.1.3.2/24' }
119          - { name: 'Ethernet2', ip: '20.1.3.2/24' }
120 ####################Postup_cislo_2_KONEC_###################################
121 ####################Postup_cislo_3_#########################################
122 #-----konfigurovani_ip_routing_na_smerovaci_Spine_1_------
123 - hosts: spine1
124   gather_facts: no
125   tasks:
126   - name: Spine1 zapnuti ip routing
127     eos_config:
128       src: ip_routing.j2
129 #-----konfigurovani_ip_routing_na_smerovaci_Spine_2_------
130 - hosts: spine2
131   gather_facts: no
132   tasks:
133   - name: Spine2 zapnuti ip routing
134     eos_config:
135       src: ip_routing.j2
136 #-----konfigurovani_ip_routing_na_smerovaci_Leaf_1_------
137 - hosts: leaf1
138   gather_facts: no
139   tasks:
140   - name: Leaf1 zapnuti ip routing
141     eos_config:
142       src: ip_routing.j2
143 #-----konfigurovani_ip_routing_na_smerovaci_Leaf_2_------
144 - hosts: leaf2
145   gather_facts: no
146   tasks:
147   - name: Leaf2 zapnuti ip routing
148     eos_config:
149       src: ip_routing.j2
150 #-----konfigurovani_ip_routing_na_smerovaci_Leaf_2_------
151 - hosts: leaf3
152   gather_facts: no
153   tasks:
154   - name: Leaf3 zapnuti ip routing
155     eos_config:
156       src: ip_routing.j2
157 ####################Postup_cislo_3_KONEC_###################################
158 ####################Postup_cislo_4_#########################################
159 #-----konfigurovani_BGP_na_smerovaci_Spine_1_------
160 - hosts: bgpspine1
161   gather_facts: no
162   vars:
163     bgpas_name: 150
164   tasks:
165    - name: Bgpspine1 konfigurovani bgp as 150
166      eos_bgp:
167       config:
168        bgp_as: "{{ vars.bgpas_name}} "
169        neighbors:
170         - neighbor: "{{ item.ip }}"
171           remote_as: "{{ item.as }}"
```

```
172         redistribute:
173           - protocol: connected
174       loop:
175           - { as: '210', ip: '10.1.1.2' }
176           - { as: '220', ip: '10.1.2.2' }
177           - { as: '230', ip: '10.1.3.2' }
178     - name: Bgpspine1 konfigurovani ECMP 2
179       eos_config:
180         src: maximum_paths.j2
181
182 #-----konfigurovani_BGP_na_smerovaci_Spine_2_------
183 - hosts: bgpspine2
184   gather_facts: no
185   vars:
186     bgpas_name: 150
187   tasks:
188     - name: Bgpspine2 konfigurovani bgp as 150
189       eos_bgp:
190        config:
191         bgp_as: "{{ vars.bgpas_name}} "
192         neighbors:
193           - neighbor: "{{ item.ip }}"
194             remote_as: "{{ item.as }}"
195         redistribute:
196           - protocol: connected
197       loop:
198           - { as: '210', ip: '20.1.1.2' }
199           - { as: '220', ip: '20.1.2.2' }
200           - { as: '230', ip: '20.1.3.2' }
201     - name: Bgpspine2 konfigurovani ECMP 2
202       eos_config:
203         src: maximum_paths.j2
204 #-----konfigurovani_BGP_na_smerovaci_Leaf1_------
205 - hosts: bgpleaf1
206   gather_facts: no
207   vars:
208     bgpas_name: 210
209   tasks:
210     - name: Bgpleaf1 konfigurovani bgp as 210
211       eos_bgp:
212        config:
213         bgp_as: "{{ vars.bgpas_name}} "
214         neighbors:
215           - neighbor: "{{ item.ip }}"
216             remote_as: "{{ item.as }}"
217         redistribute:
218           - protocol: connected
219       loop:
220           - { as: '150', ip: '10.1.1.1' }
221           - { as: '150', ip: '20.1.1.1' }
222     - name: Bgpleaf1 konfigurovani ECMP 2
223       eos_config:
224         src: maximum_paths.j2
225 #-----konfigurovani_BGP_na_smerovaci_Leaf2_------
226 - hosts: bgpleaf2
227   gather_facts: no
228   vars:
229     bgpas_name: 220
230   tasks:
```

```
231     - name: Bgpleaf2 konfigurovani bgp as 220
232       eos_bgp:
233        config:
234         bgp_as: "{{ vars.bgpas_name}} "
235         neighbors:
236          - neighbor: "{{ item.ip }}"
237            remote_as: "{{ item.as }}"
238         redistribute:
239          - protocol: connected
240       loop:
241          - { as: '150', ip: '10.1.2.1' }
242          - { as: '150', ip: '20.1.2.1' }
243     - name: Bgpleaf2 konfigurovani ECMP 2
244       eos_config:
245         src: maximum_paths.j2
246 #-----konfigurovani_BGP_na_smerovaci_Leaf3_------
247 - hosts: bgpleaf3
248   gather_facts: no
249   vars:
250     bgpas_name: 230
251   tasks:
252    - name: Bgpleaf3 konfigurovani bgp as 230
253      eos_bgp:
254       config:
255        bgp_as: "{{ vars.bgpas_name}} "
256        neighbors:
257         - neighbor: "{{ item.ip }}"
258           remote_as: "{{ item.as }}"
259        redistribute:
260         - protocol: connected
261      loop:
262         - { as: '150', ip: '10.1.3.1' }
263         - { as: '150', ip: '20.1.3.1' }
264     - name: Bgpleaf3 konfigurovani ECMP 2
265       eos_config:
266         src: maximum_paths.j2
267 ####################Postup_cislo_4_KONEC_####################################
268 ####################Postup_cislo_5_#########################################
269 #-----konfigurovani_Vlan_30_40_na_smerovaci_Leaf_1_------
270 - hosts: leaf1
271   gather_facts: no
272   tasks:
273    - name: Vytvoreni Vlan 30,40 a pridani do Ethernet 3-4
274      eos_vlan:
275        vlan_id: "{{ item.ad }}"
276        state: present
277        interfaces:
278         - "{{ item.eth }}"
279      loop:
280         - { eth: 'Ethernet3', ad: '30' }
281         - { eth: 'Ethernet4', ad: '40' }
282 #-----konfigurovani_Vlan_30_40_na_smerovaci_Leaf_2_------
283 - hosts: leaf2
284   gather_facts: no
285   tasks:
286    - name: Vytvoreni Vlan 30,40 a pridani do Ethernet 3-4
287      eos_vlan:
288        vlan_id: "{{ item.ad }}"
289        state: present
```

```
290         interfaces:
291           - "{{ item.eth }}"
292       loop:
293           - { eth: 'Ethernet3', ad: '30' }
294           - { eth: 'Ethernet4', ad: '40' }
295 #-----konfigurovani_Vlan_30_40_na_smerovaci_Leaf_3_------
296 - hosts: leaf3
297   gather_facts: no
298   tasks:
299    - name: Vytvoreni Vlan 30,40 a pridani do Ethernet 5
300      eos_vlan:
301        vlan_id: "{{ item.ad }}"
302        state: present
303        interfaces:
304          - "{{ item.eth }}"
305      loop:
306          - { eth: 'Ethernet5', ad: '50' }
307 ###################Postup_cislo_5_KONEC_#####################################
308 ###################Postup_cislo_6_###########################################
309 #-----konfigurovani_IP_adres_Vlans_30_40_na_smerovaci_Leaf_1_------
310 - hosts: leaf1
311   gather_facts: no
312   tasks:
313    - name: Konfigurovani Ip adres Vlan 30,40
314      eos_l3_interface:
315       name: "{{ item.name }}"
316       ipv4: "{{ item.ip }}"
317      loop:
318          - { name: 'Vlan30', ip: '192.168.30.254/24' }
319          - { name: 'Vlan40', ip: '192.168.40.254/24' }
320 #-----konfigurovani_IP_adres_Vlans_30_40_na_smerovaci_Leaf_2_------
321 - hosts: leaf2
322   gather_facts: no
323   tasks:
324    - name: Konfigurovani Ip adres Vlan 30,40
325      eos_l3_interface:
326       name: "{{ item.name }}"
327       ipv4: "{{ item.ip }}"
328      loop:
329          - { name: 'Vlan30', ip: '192.168.30.254/24' }
330          - { name: 'Vlan40', ip: '192.168.40.254/24' }
331 #-----konfigurovani_IP_adres_Vlans_30_40_na_smerovaci_Leaf_3_------
332 - hosts: leaf3
333   gather_facts: no
334   tasks:
335    - name: Konfigurovani Ip adres Vlan 50
336      eos_l3_interface:
337       name: "{{ item.name }}"
338       ipv4: "{{ item.ip }}"
339      loop:
340          - { name: 'Vlan50', ip: '192.168.50.254/24' }
341 ###################Postup_cislo_6_KONEC_#####################################
342 ###################Postup_cislo_7_###########################################
343 #-----konfigurovani_IP_adres_loopback1_na_smerovaci_Leaf_1_------
344 - hosts: leaf1
345   gather_facts: no
346   tasks:
347    - name: Konfigurovani loopback1 a jeho Ip adresy
348      eos_l3_interface:
```

```
349       name: "{{ item.name }}"
350       ipv4: "{{ item.ip }}"
351      loop:
352         - { name: 'loopback1', ip: '110.110.110.1/24' }
353 #-----konfigurovani_IP_adres_loopback1_na_smerovaci_Leaf_2_------
354 - hosts: leaf2
355   gather_facts: no
356   tasks:
357    - name: Konfigurovani loopback1 a jeho Ip adresy
358      eos_l3_interface:
359       name: "{{ item.name }}"
360       ipv4: "{{ item.ip }}"
361      loop:
362         - { name: 'loopback1', ip: '120.120.120.1/24' }
363 ####################Postup_cislo_7_KONEC_###################################
364 ####################Postup_cislo_8_########################################
365 #-----konfigurovani_VXLAN_na_smerovaci_Leaf_1_------
366 - hosts: leaf1
367   gather_facts: no
368   vars:
369     vlan_name: VXLAN1
370     vlan1_id: 30
371     vlan2_id: 40
372     vtep_ip: 120.120.120.1
373   tasks:
374   - name: Konfigurovani VXLAN1
375     eos_config:
376       src: l2vxlan.j2
377 #-----konfigurovani_VXLAN_na_smerovaci_Leaf_2_------
378 - hosts: leaf2
379   gather_facts: no
380   vars:
381     vlan_name: Konfigurovani VXLAN1
382     vlan1_id: 30
383     vlan2_id: 40
384     vtep_ip: 110.110.110.1
385   tasks:
386   - name: Configure L2 VXLAN
387     eos_config:
388       src: l2vxlan.j2
389 ####################Postup_cislo_8_KONEC_###################################
```

Výpis 4.16: Příklad skriptu pro Ansible pro laboratorní úlohu číslo 2.

4.4.1 Odpovědi na kontrolní otázky

- Na které vrstvě existuje v úloze VXLAN? Vrstva číslo 3.
- Jaký protokol byl využit pro předání směrovacích informací mezi směrovači? BGP.
- V jaké úloze byl vyvolán modul pomocí SSH? Úloha číslo 2 modul eos_bgp.

Závěr

V rámci této diplomové práce byl zhodnocen současný vývoj v oblasti SDN a možnosti programovatelnosti SDN prvků pomocí API rozhraní. Pro lepší pochopení, možnosti programovatelnosti za pomoci SDN API byly vytvořeny laboratorní úlohy, pomocí kterých se dá vyzkoušet a osvětlit programovatelnost SDN API.

V teoretické části byla popsána základní architektura SDN a provoz uvnitř SDN mezi jeho jednotlivými vrstvami. Vylíčeny byly i protokoly komunikace „Southbound“ rozhraní a „Northbound“ rozhraní. Laboratorní úlohy se tyká „Northbound“ rozhraní, protože toto rozhraní nebylo dodnes standardizováno. Protokol REST je v současnosti nejčastěji používaným rozhraním a většina kontrolérů ho implementuje. Následně byly označeny výhody a nevýhody využívání SDN vytvořených na základě Open SDN, existujících API a superponovaných sítí a hypervisorů. Byla zde popsána programovatelnost SDN API a typické příklady protokolů SDN a API. Posledním bodem teoretické části bylo zhodnocení současného vývoje v oblasti SDN. Podle aktuální zprávy analýzy trhu, kterou připravila společnost Data Bridge Market Research, se očekává, že globální trh SDN do roku 2026 vzroste na odhadovanou hodnotu 67,98 miliard USD. Následná část práce obsahuje popis nejvýznamnějších poskytovatelů a řešení SDN, jako jsou Cisco ACI, VMware NSX, Nokia Nuage Networks, Juniper's Contrail Network, Arista Networks, Huawei atd. Z analýzy vyplývá, že se SDN stále nachází ve fázi aktivního vývoje. Následující vývojové trendy budou patřit vývoji SDN v oblasti páteřních a distribučních sítí, což je SDN mimo datová centra.

V praktické části, jak již bylo zmíněno, byly vytvořeny laboratorní úlohy, které se zabývají programováním SDN API.

V první laboratorní úloze bylo využito hardwarové řešení od společnosti F5 Networks BIG-IP a nakonfigurován BIG-IP loadbalancer pro vyrovnání zátěže serverů přes API. Úloha obsahuje teoretický úvod a pracovní postup. V teoretickém úvodu je popsána architektura SDN, REST API a BIG-IP. Ve výsledku přináší laboratorní úloha pochopení toho, jak jednoduše může být nakonfigurován velký počet síťových prvků a jaké výhody SDN API přináší pro aplikace na příkladu BIG-IP.

Druhá laboratorní úloha používá Arista EOS od společnosti Arista Network. Pomocí virtuálních směrovačů vEOS-lab je potřeba nakonfigurovat přes eAPI „Leaf-Spine“ Layer 3/ECMP síť. Poté je potřeba v této síti zprovoznit Vxlan a ověřit komunikace mezi „overlay“ a „underlay“ sítěmi. Úloha obsahuje teoretický úvod a pracovní postup všech úkolů. V teoretickém úvodu je popsána architektura SDN, eAPI, „Leaf-Spine“ sít, BGP, Vxlan atd. Pomocí laboratorní úlohy lze pochopit, jak jednoduše může být nakonfigurován prostý data centr, nebo velký počet síťových prvků na příkladu Arista EOS a jaké výhody SDN API přináší.

Literatura

[1] Olifer, V.; Olifer N. *Computer networks. Principles, technologies, protocols*, Sankt-Peterburg: Piter, 2020. ISBN 978-5-4461-1426-9.

[2] Taheri, Javid. *Big Data and Software Defined Networks*, London: Institution of Engineering and Technology, 2018. 978-1-78561-305-0.

[3] Huang, Dijiang. aj. *Software-Defined Networking and Security*, Boca Raton: CRC Press, 2019. ISBN 978-0-8153-8114-3.

[4] Coker, Oswald. aj. *Software-Defined Networking with OpenFlow, Second Edition*, Birmingham: Packt Publishing Ltd., 2017. ISBN 978-1-78398-428-2.

[5] Green, Todd. Software Defined Networks *A Comprehensive Approach. Second Edition*, India: SPi Global, 2017. 978-0-12-804555-8.

[6] Medjaoui, M. aj. *Continuous API Managemen*, Sankt-Peterburg: Piter, 2020. ISBN 978-5-4461-1232-6.

[7] Allamaraju, S. *RESTful Web Services Cookbook*, Sebastopol: O'Reilly Media, 2010. ISBN 978-0-596-80168-7.

[8] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*, Irvine, 2000, Doctoral dissertation, University of California.

[9] Richardson, L. aj. *RESTful Web APIs*, Sebastopol: O'Reilly Media, 2017. ISBN 978-1-4493-5806-8.

[10] Edelman, J. aj. *Network Programmability and Automation*, Moscow: DMK Press, 2019. ISBN 978-5-97060-699-5.

[11] Hochstein, Lorin.; Moser, Rene. *AnsibLeUp & Running*, Moscow: DMK Press, 2018. ISBN 978-5-97060-513-4.

[12] Chou, Eric. aj. *Mastering Python Networking*, Birmingham: Packt Publishing Ltd., 2017. ISBN 978-1-78439-700-5.

[13] Data Bridge Market Research. *Global SDN Market – Industry Trends and Forecast to 2026* [online]. 2019 [cit. 1. 4. 2020]. Dostupné na internetu: <`https://www.databridgemarketresearch.com/reports/global-sdn-market`>.

[14] Gartner. *Magic Quadrant for Data Center Networking* [online]. 15.7.2019 [cit. 1. 4. 2020]. Dostupné na internetu: <https://www.gartner.com/doc/reprints?id=1-1OAIHHDJ&ct=190718&st=sb>.

[15] The Forrester Wave. *Hardware Platforms For Software-Defined Networking, Q1 2018* [online]. 2018 [cit. 1. 4. 2020]. Dostupné na internetu: <https://reprints.forrester.com/#/assets/2/327/RES136622/reports>.

[16] IDC. *SD-WAN Infrastructure Market Poised to Reach $5.25 Billion in 2023* [online]. 2019 [cit. 1. 4. 2020]. Dostupné na internetu: <https://www.idc.com/getdoc.jsp?containerId=prUS45380319>.

[17] F5 Networks. *F5 Python SDK Documentation* [online]. 2017 [cit. 1. 4. 2020]. Dostupné na internetu: <https://f5-sdk.readthedocs.io/en/latest/>.

[18] Red Hat, Inc. *YAML Syntax* [online]. 2020 [cit. 1. 4. 2020]. Dostupné na internetu: <https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html>.

[19] Red Hat, Inc. *All modules* [online]. 2020 [cit. 1. 4. 2020]. Dostupné na internetu: <https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html>.

[20] Arista EOS+ CS. *Python Client for eAPI* [online]. 2015 [cit. 1. 4. 2020]. Dostupné na internetu: <https://pyeapi.readthedocs.io/en/master/modules.html>.

[21] Red Hat, Inc. *Network modules* [online]. 2020 [cit. 1. 4. 2020]. Dostupné na internetu: <https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html#eos>.

Seznam symbolů, veličin a zkratek

ACI	Application Centric Infrastructure
API	Application Programming Interface
APIC	Application Policy Infrastructure Controller
AS	Autonomous System
ASIC	Application-specific Integrated Circuit
BCF	Big Cloud Fabric
BGP	Border Gateway Protocol
CAGR	Compound Annual Growth Rate
CLI	Command Line Interface
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DSL	Domain-specific Language
ECMP	Equal-Cost Multi-Path Routing
EIGRP	Enhanced Interior Gateway Routing Protocol
EOS	Extensible Operating System
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HD	High-definition
HTTP	Hypertext Transfer Protocol
IDC	International Data Corporation
IoT	Internet of Things
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
JSON	JavaScript Object Notation
MAC	Media Access Control
MLAG	Multi-Chassis Link Aggregation
NetConf	Network Configuration Protocol
NFV	Network Function Virtualization
NOS	Network Operating System
ONOS	Open Network Operating System
ONF	Open Networking Foundation
OVSDB	Open vSwitch Database
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PAD	Programmable Abstraction of Datapath
QoS	Quality of Service

REST	Representational State Transfer
ROFL	Revised OpenFlow Library
RPC	Remote Procedure Call
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
SSH	Secure Shell
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VPC	Virtual Personal Computer
VPN	Virtual Private Network
VNI	Virtual Network Identifier
VRF	Virtual Routing and Forwarding
VRS	Verification Router Service
VSP	Virtualized Services Platform
VSC	Virtualized Services Controller
VSD	Virtualized Services Directory
VTEP	Vitual Tunnel End Point
VXLAN	Virtual Extensible Local Area Network
SDK	Software Development Kit
SD-WAN	Software Defined Networking in a Wide Area Network
SOAP	Simple Object Access Protocol
WAN	Wide Area Networking
XML	Extensible Markup Language
YAML	Ain't Markup Language
5G	Fifth Generation

Seznam příloh

A Nastavení aplikace Vmplayer pro 1. a 2. laboratorní úlohu

Tato příloha popisuje, jak správně zapojit virtuální servery. Servery se dají spustit pomocí Vmplayer verze 14.x nebo 15.x. Nastavení síťových adaptérů v laboratorní úloze číslo 1, viz tabulka A.1. Tabulka A.2 popisuje nastavení síťových adaptérů laboratorní úlohy číslo 2.

Tab. A.1: Nastavení síťových adaptérů v laboratorní úloze číslo 1

Název serveru	Název síťového adaptéru	IP adres	Druh ip dresy
LAB_Nginx	Network Adapter	10.0.0.65/24	static
LAB_Nginx_Ansible	Network Adapter	10.0.0.60/24	static
	Network Adapter 2	172.25.35.0/24	dhcp

Tab. A.2: Nastavení síťových adaptérů v laboratorní úloze číslo 2

Název serveru	Název síťového adaptéru	IP adres	Druh ip dresy
Debian 10 Ansibl	Network Adapter	172.25.35.0/24	dhcp

B Obsah přiloženého DVD

- `/` ..... Kořenový adresář DVD
  - `VM`. ..... Soubor disků virtuálních serverů
    - `LAB1` ..... Soubor disků virtuálních serverů pro laboratorní úlohu 1
      - `LAB_Nginx`. ..... Disk virtuálního serveru ve formátu ZIP
      - `LAB_Nginx_Ansible`. ..... Disk virtuálního serveru ve formátu ZIP
    - `LAB2` ..... Soubor disků virtuálních serverů pro laboratorní úlohu 2
      - `Debian_10_Ansible`. ..... Disk virtuálního serveru ve formátu ZIP
  - `Diplom./` ..... Diplomová práce ve formátu PDF
  - `Ansible./` ..... Prostředí Ansible pro laboratorní úlohu 2 ve formátu ZIP
  - `t_script_Python./` . Tělo skriptu Python pro laboratorní úlohu 2 ve formátu PY
  - `t_script_Ansible./` .... Tělo skriptu Ansible pro laboratorní úlohu 2 ve formátu YML
  - `Scripts./` ..... Skripty pro vyučující do laboratorních úloh 1 a 2 ve formátu ZIP
  - `EVE-NG_LAB./` ..... Topologie sítě pro EVE-NG ve formátu ZIP