SOFTVERSKI DEFINIRANE MREŽE U OPENSTACK OBLAČNOM RAČUNARSTVU

Nemanić, Antun

Master's thesis / Specijalistički diplomski stručni

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:775632

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-27**

ALGEBRA SVEUČILIŠTE

Repository / Repozitorij:

Algebra Univerity - Repository of Algebra Univerity

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VISOKO UČILIŠTE ALGEBRA

DIPLOMSKI RAD

SOFTVERSKI DEFINIRANE MREŽE U OPENSTACK OBLAČNOM RAČUNARSTVU

Antun Nemanić

Zagreb, rujan 2019.

Predgovor

Mojoj majci.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme diplomskog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

U ovome radu obrađena je tema softverski definiranih mreža i kako se one uklapaju u OpenStack oblačno računarstvu. Obrađen je koncept softverski definiranih računalnih mreža, osnovni protokoli poput VXLAN-a, NVGRE-a i GENEVE. U laboratorijskom dijelu napravljena je instalacija OpenStack okruženja s naglaskom na mrežni dio, Neutron. Napravljena je integracija i interoperabilnost Linux Bridge i Open vSwitch mrežnog adaptera i analiza VXLAN komunikacije između ta dva tipa softverskih mrežnih adaptera. Zaključak je da je OpenStack vrlo modularno i cjelovito cloud okruženje s mogućnostima proširenja i podrške *open source* zajednice, a upravljanje softverskim mreža funkcionira vrlo efikasno i jednostavno.

Ključne riječi: OpenStack, cloud, računalne mreže, SDN, softverski definirane mreže, VXLAN, NVGRE, GENEVE , Neutron, Open vSwitch

In this master thesis I will go through process of explaining Software Defined Networks in computing and their implementation in OpenStack cloud computing. For start, terms SDN, protocols VXLAN, NVGRE and GENEVE will be explained. In laboratory part of this thesis I will install OpenStack Cloud with accent on network part, Neutron. Linux Bridge and Open vSwitch interoperability and integration has been implemented. After analysing VXLAN communication, conclusion is that OpenStack is very modular and whole solution with extension possibilities and open-source community, and administration of software defined network is very effective and simple.

Key words: OpenStack, cloud, computing networks, SDN, Software Defined Networks, VXLAN, NVGRE, GENEVE , Neutron, Open vSwitch

Sadržaj

1. Uvod 1

2. Softverski definirane mreže (SDN) i Virtualizacija mrežnih uloga (NFV) 3

2.1. Motiv i povijest 4

2.2. Protokoli SDN-a – VXLAN, NVGRE, GENEVE 7

2.2.1. VXLAN 7

2.2.2. NVGRE 9

2.2.3. STT 11

2.2.4. GENEVE 11

2.3. Postojeća SDN rješenja 12

2.3.1. Cisco ACI 12

2.3.2. VMware NSX 14

2.3.3. Usporedba 14

2.4. Network Function Virtualization - NFV 17

2.4.1. Open vSwitch 17

3. OpenStack 19

3.1. OpenStack komponente 20

3.2. OpenStack mreža – Neutron 24

3.2.1. Mogućnosti Neutrona 25

3.2.2. Tipovi mreža u Neutronu 28

3.2.3. ML2 plugin 32

3.2.4. Network Namespace 34

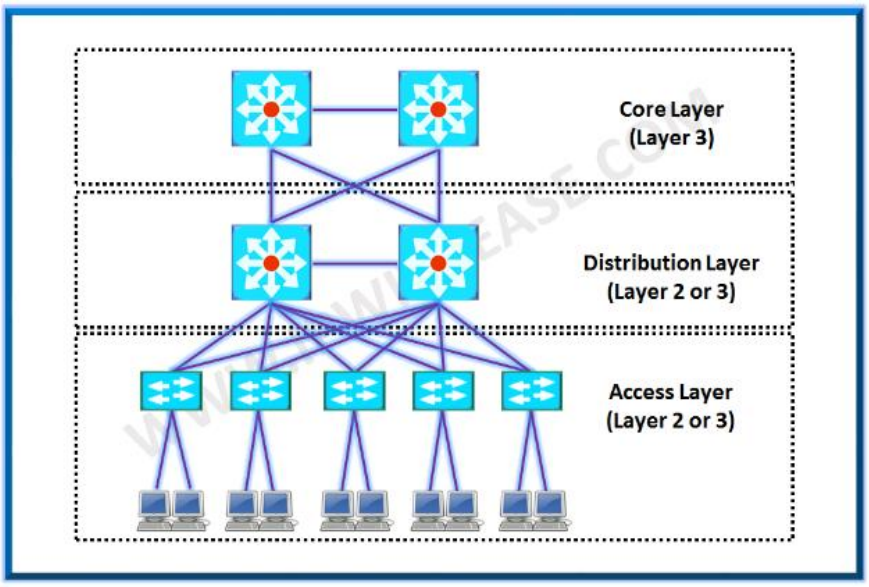
4. Implementacija OpenStack okruženja 36

4.1. Osnovna konfiguracija 38

4.2. Instalacija Neutrona i mrežna konfiguracija 48
4.3. Testiranje i mjerenja 56
4.4. OpenStack vs VMware vRealize Automation 60
Zaključak 63
Popis kratica 65
Popis slika 66
Popis tablica 68
Literatura 69

1. Uvod

Software Defined Networking, SDN, je potpuno nova paradigma u računalnim mrežama. Pristup računalnim mrežama, konfiguraciji uređaja i arhitekturi nije se mijenjao od osamdesetih godina prošlog stoljeća. Već svima u IT-u dobro poznata troslojna arhitektura (sloj jezgre, distribucije i pristupni sloj), Slika 1, ima svoje prednosti, ali dovodi do vrlo zahtjevne administracije velikog broja uređaja, spajanje na svaki kod promjene konfiguracije kroz komadnu liniju i održavanje svakog pojedinačnog uređaja. SDN uvodi potpuno novi pristup u kojemu centralni softver upravlja cijelom mrežom, bez potrebe za pojedinačnom konfiguracijom svakog uređaja. SDN softver na sebe preuzima kontrolnu razinu koja se do sada nalazila na svakom pojedinačnom uređaju čineći time mrežne uređaje pod sobom jednostavnim „radnicima".



Slika 1[1] Staromodna mrežna arhitektura

OpenStack je oblačni operacijski sustav osmišljen kako bi kontrolirao velike skupine računalnih, podatkovnih i mrežnih resursa u jednom računalnom centru (*datacenter, engl.*). Zanimljivo za napomenuti je da je OpenStack sustav otvorenog računarskog koda dostupan svima, i kao takav je izvrstan temelj za edukacijsko-znanstvenu zajednicu i demonstriranje raznih funkcionalnosti. U svom radu fokusirati ću se na softverski definirane mreže. Cilj mi

[1] https://ipwithease.com/3-layer-cisco-hierarchical-model/

je pokazati što su softverski definirane računalne mreže, koji je njihov benefit, i kako će izgledati računalne mreže u vrlo skoroj budućnosti. Demonstracija je u sklopu integracije s virtualizacijskim sustavima ispod OpenStack orkestratora. Koncentracija će biti na Neutron, mrežnu komponentu OpenStack sustava koju ću pobliže opisati i objasniti način funkcioniranja u mrežnom sustavu.

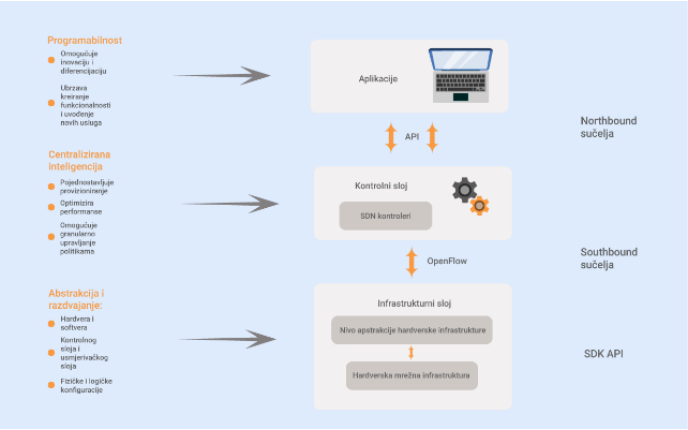
2. Softverski definirane mreže (SDN) i Virtualizacija mrežnih uloga (NFV)

SDN je novi koncept softverskog upravljanja mrežom. Definiciju SDN-a napravila je Open Networking fundacija koja aktivno sudjeluje u razvoju SDN tehnologija. Prema njihovoj definiciji, „SDN razdvaja kontrolni od podatkovnog sloja“[2] na način da središnji kontroleri imaju apstraktan pogled na mrežu, Slika 2. Karakteristike SDN-a:

- Automatizacija
- Orkestracija
- Skalabilnost
- Mikrosegmentacija

SDN arhitektura sastoji se od:

1. Infrastrukturnog sloja koji obuhvaća mrežne uređaje koji služe za usmjeravanje i preklapanje prometa po informacijama i zahtjevima koje dobije od kontrolnog sloja.
2. Kontrolnog sloja koji upravlja i kontrolira mrežne uređaje u infrastrukturnom sloju. Konceptualna ideja ovog sloja je da ne ovisi o proizvođaču uređaja na infrastrukturnom sloju.
3. Aplikativnog sloja koji služi za komunikaciju aplikacija sa zahtjevima o mrežnim funkcionalnostima prema kontroleru na kontrolnom sloju.



Slika 2 SDN arhitektura[2]

[2] https://www.ictbusiness.info/kolumne/upoznajte-glavne-prednosti-sdn-tehnologija

2.1. Motiv i povijest

Sami početci računalnih mreža datiraju od šezdesetih godina prošlog stoljeća kada je ARPA (Advanced Research Projects Agency) izumila ARPANET mrežu za komunikaciju između uređaja na vlastitoj mreži. Prvi mrežni uređaj naziva se *Interface Message Processor*, IMP, Slika 3.

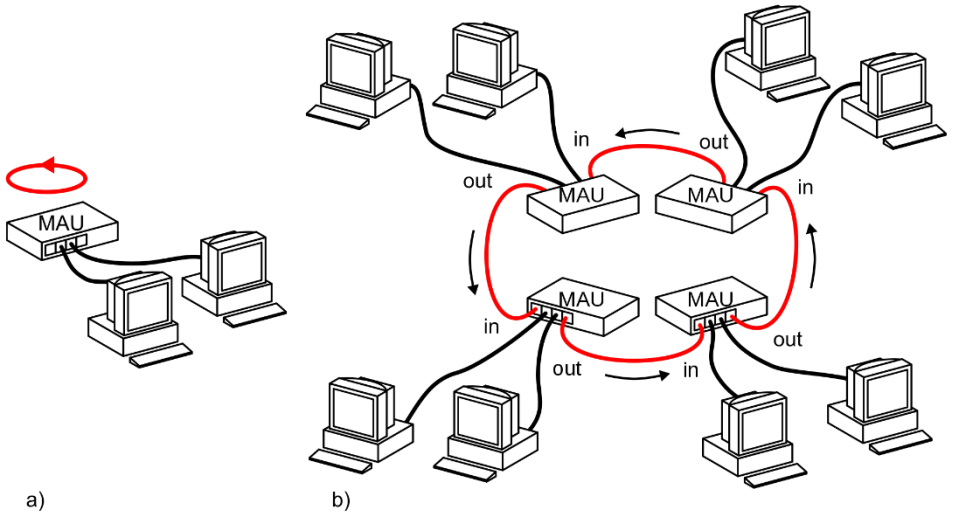
Slika 3[3] IMP

Protokol kojim su uređaji komunicirali nazivao se *Network Control Protocol*, NCP.

U početcima računalnih mreža koristio se „Token ring“ kao LAN (*Local Area Network,* engl.) protokol za međusobnu komunikaciju uređaja na lokalnoj mreži. Uređaji su spojeni u krug i među njima „kruži“ token koji dopušta komunikaciju na mreži uređaju koji je u

[3] https://www.net.t-labs.tu-berlin.de/teaching/computer_networking/01.09.htm

posjedu tokena. Centralni uređaj je MAU (*Multistation Access Unit*) na kojeg se spajaju uređaji i koji služi za prosljeđivanje tokena i paketa mrežom.



Slika 4 Token ring[4]

Ovakav sustav se nakon nekog vremena pokazao nedovoljno efikasnim, a Ethernet je, pogotovo pojavom preklopnika preuzeo primat u LAN tehnologijama.

Ethernet je sustav, skupina protokola za povezivanje uređaja na LAN mreži. Koristi CSMA/CD protokol za detekciju kolizije na mreži (*carrier-sense multiple access with collision detection*, engl.). Prije nego što sustav počne komunicirati, napravi provjeri komunicira li itko drugi mreži. U početku je brzina Etherneta bila 10 Mbit/s, ali brzo je standard dostigao 100Mbit/s dok je danas 1Gbit/s uobičajena brzina Ethernet linkova, sve do 100Gbit/s. Sljedeća Slika 5 prikazuje sadržaj Ethernet okvira:

Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets

Slika 5 Ethernet okvir[5]

- Preambula i SFD – za sinkronizaciju između uređaja, najava dolaska paketa
- MAC *destination* – Fizička adresa mrežne kartice na koju se šalje paket
- MAC *source* – Adresa mrežne kartice s koje dolazi paket

[4] https://en.wikipedia.org/wiki/Token_ring#/media/File:Token_ring.svg
[5] https://clinetworking.com/2018/04/29/2-2-interpret-ethernet-frame-format/

- 802.1Q tag – VLAN tag za razlikovanje enkapsulacije na preklopniku (kasnije u radu)
- *Ethertype or length* – dužina okvira i tip protokola u okviru
- *Payload* – „korisni“ podatci, IP paket
- FCS – CRC provjera paketa na greške u prijenosu paketa

Rastom i razvojem LAN mreža dolazi do potrebe za povezivanjem udaljenih mreža. Dolazi do razvoja VLAN-a, *Virtual Local Area Network*. VLAN tag unutar *ethernet* okvira omogućuje preklopniku da odvoji *broadcast* domene jednu od druge, brže preklopi *ethernet* okvire koji se ne nalaze kod njega (prema VLAN tagu). Osim toga, donosi vrlo efikasnu logičku podjelu preklopnika na više logičkih cjelina time više *broadcast* domena. To zahtjeva konfiguraciju VLAN-a na svakom preklopniku na putu VLAN paketa.

802.1q standard je dizajniran za podršku VLAN-ova na Ethernet mreži. Standard definira sustav VLAN tagiranja na *ethernet* mreži.[6] U prijašnjoj tablici imali smo prikaz Ethernet okvira na kojem vidimo 802.1Q zaglavlje. Ono se sastoji od sljedećih polja, Slika 6:

<table>
<tr><th>16 bits</th><th>3 bits</th><th>1 bit</th><th>12 bits</th></tr>
<tr><td rowspan="2">TPID</td><td colspan="3">TCI</td></tr>
<tr><td>PCP</td><td>DEI</td><td>VID</td></tr>
</table>

Slika 6 802.1q VLAN tag[7]

Gdje TPID označava da je *ethernet frame* tagiran VLAN tagom. PCP polje služi za QOS na Ethernet razini, DEI za kontrolu zagušenja, a VID, VLAN ID, nama najzanimljivije polje je 12 bitno polje koje sadrži VLAN broj za identifikaciju VLAN-a kojemu pripada ethernet okvir. Binarnom matematikom 2^{12} dolazimo do brojke od 4096 mogućih VLAN-ova koje preklopnici mogu razlikovati.

Na ovome sloju koristi se i STP (*Spanning Tree Protocol*, engl.) za sprečavanje petlji pošto Ethernet okvir nema TTL polje. Ukratko, STP blokira puteve koji nisu najbolji uzimavši u obzir centralni (*root*, engl.) preklopnik s ciljem da imamo aktivnu samo najbolju putanju od

[6] https://ieeexplore.ieee.org/document/6991462
[7] https://techtalks101.wordpress.com/2016/02/09/vlan-tagging/

centralnog do rubnog preklopnika. To znači da ne iskorištavamo sve veze između preklopnika nego koristimo samo jednu putanju, a ostale su rezervne i postaju aktivne u slučaju kada glavni STP put više ne bude dostupan.

S porastom virtualizacije i oblačnog računarstva dolazi do potrebe za promjenom pristupa računalnim mrežama. 4096 VLAN-ova nije dovoljno za *cloud* infrastrukturu, dolazi do potrebe za preklapanjem adresnih prostora i dolazi do razvoja novih rješenja.[8] VXLAN (*Virtual eXtensible LAN*), NVGRE i GENEVE su postojeća rješenja koja ću pobliže objasniti u sljedećim poglavljima.

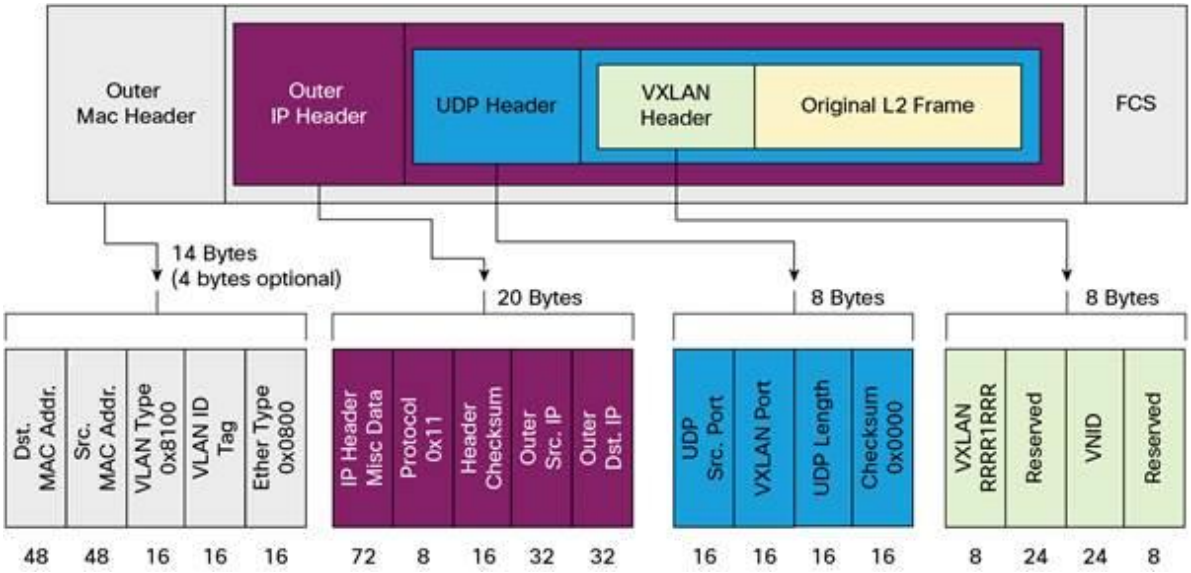
2.2. Protokoli SDN-a – VXLAN, NVGRE, GENEVE

Pojava ovih protokola uzrokovana je željom da se riješe problemi staromodne mreže. Virtualizacija mreže težnja je modernog računarstva kako bi se mogle postići kompleksnije stvari uz što manje razmišljanja o mreži kao temelju komunikacije između sustava. Protokoli su skalabilni i rješavaju probleme malo broja VLAN-ova i preklapanja mreža tuneliranjem.

2.2.1. VXLAN

Virtual Extensible LAN je enkapsulacijski protokol koji adresira i rješava probleme klasičnog VLAN-a na način da enkapsulira ethernet okvire u UDP *datagrame*. VXLAN krajnje točke (*endpointi*, engl.) zatvaraju *VXLAN Tunnel Endpoint* (VTEP) L3 tunele putem kojih prenose VXLAN. Svaki VXLAN segment je identificiran 24 bitnim ID-em, *VXLAN Network Identifier* (VNI) što omogućava 16 milijuna VXLAN nepreklapajućih segmenata, Slika 7.

[8] https://tools.ietf.org/html/rfc7348#page-5

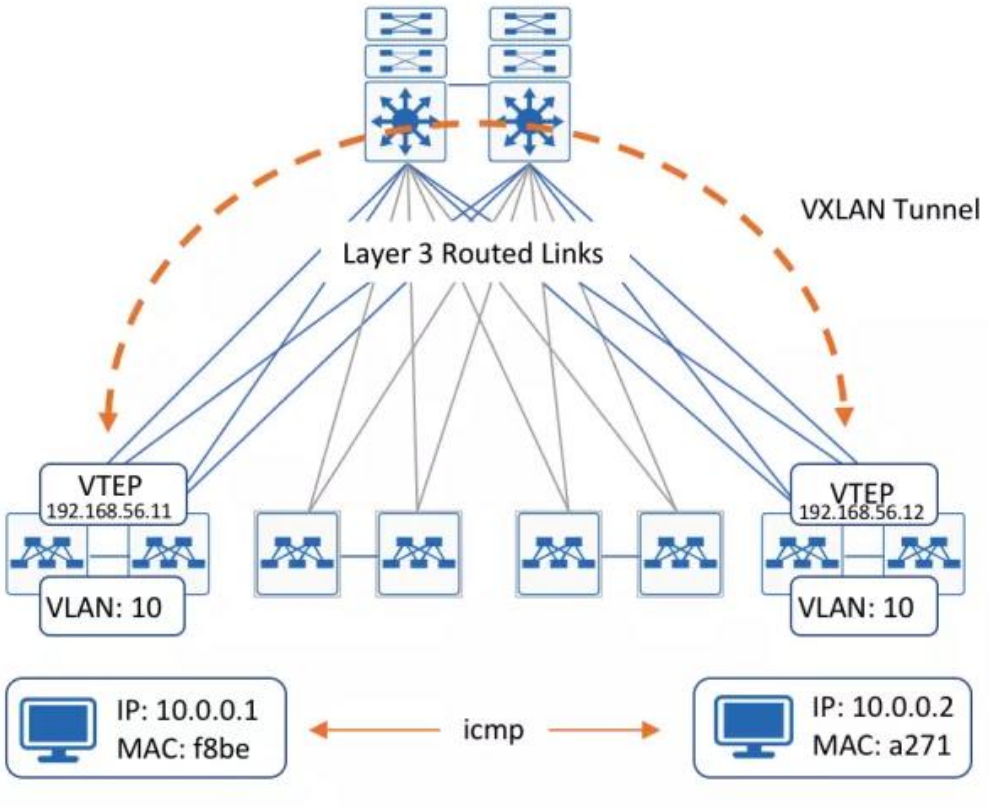


Slika 7 VXLAN okvir[9]

Kako VXLAN koristi IP mrežu za prijenos paketa (*Underlay network*, engl.) to čini VXLAN *overlay* tehnologiju potpuno neovisnu o transportnom sloju. Kreiraju se VTEP tuneli kojima nije bitno kojim putem se prenose podatci u mreži ispod njih, usporedivo s IPsec VPN tunelima (izvan opsega ovog rada). Ovo čini skaliranje jednostavnijim jer se *underlay* mreža može mijenjati i nadograđivati bez utjecaja na VXLAN. Ovdje se najčešće koristi „*Spine/Leaf*“ *underlay* IP mreža, o čemu će biti govora nešto kasnije.[10]

[9] https://www.cisco.com/c/dam/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.doc/_jcr_content/renditions/white-paper-c11-729383_1.jpg
[10] https://networkdirection.net/articles/routingandswitching/vxlanoverview/

Slika 8 prikazuje kako funkcioniraju VXLAN tuneli.



Slika 8 VXLAN tunel[11]

Host 10.0.0.1 inicira ICMP paket prema 10.0.0.2. VXLAN *switch* prima paket, gleda u MAC tablicu gdje mu piše da se *destination* MAC adresa nalazi na VTEP sučelju. *Switch* enkapsulira IP paket dodajući mu VXLAN zaglavlje i šalje tunelirani paket na 192.168.56.12. VTEP destinacijski *switch* skida VLAN zaglavlje gleda MAC adresu i prosljeđuje paket na sučelje na kojem se nalazi *host* 10.0.0.2 s pripadajućom MAC adresom.

2.2.2. NVGRE

Network Virtualization using Generic Routing Encapsulation (NVGRE) je također protokol za enkapsulaciju i tuneliranje VLAN-ova sa zadaćom rješavanja postojećih, već spomenutih ograničenja. Glavna karakteristika je korištenje 24 bitnog *Tenant Network Indentifier* polja

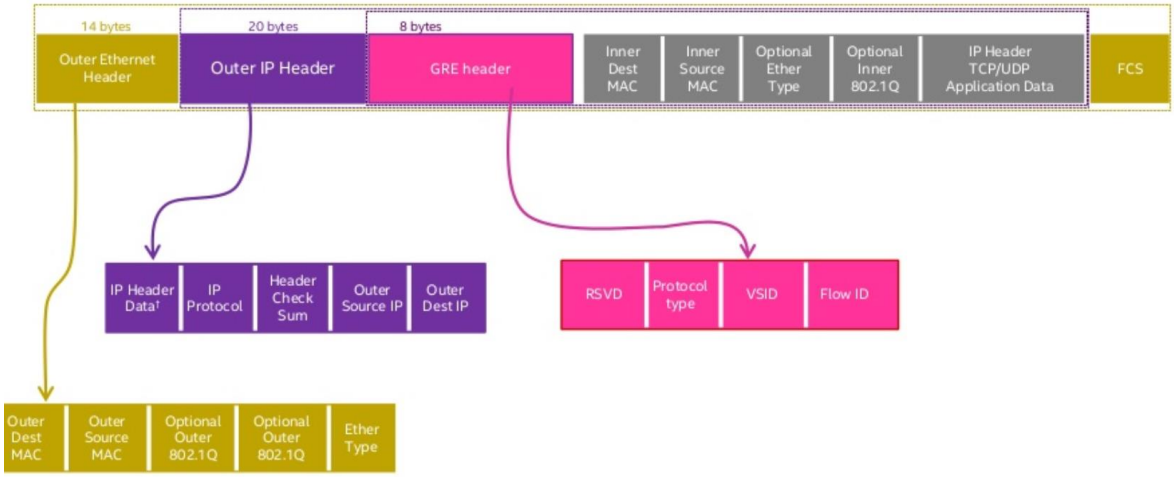
[11] https://overlaid.net/2018/08/27/arista-bgp-evpn-overview-and-concepts/

(TNI) putem GRE protokola za kreiranje izoliranih L2 mreža koje mogu biti rastegnute između lokacija.[12] Slika 9

Generic Routing Encapsulation (GRE) je protokol za tuneliranje koji stvara virtualne *point-to-point* linkove kako bi se razni drugi mrežni protokoli (npr. *routing*) povezali s udaljenim lokacijama putem IP mreže. GRE endpoint enkapsulira *payload,* koji se usmjerava po IP mreži dok ne stigne do GRE destinacije koja dekapsulira GRE paket i prosljeđuje ga na krajnji cilj.[13]

NVGRE postiže isti cilj kao i VXLAN, ali koristi drugačiju tehnologiju enkapsuliranja paketa. GRE je protokol podržan od strane velikog broja uređaja što teoretski znači da NVGRE „out of the box" ima podršku na većem broju uređaja od VXLAN-a.

Slika 9 prikazuje NVGRE okvir



Slika 9 NVGRE okvir[14]

[12] https://searchnetworking.techtarget.com/definition/NVGRE-Network-Virtualization-using-Generic-Routing-Encapsulation

[13] https://searchnetworking.techtarget.com/definition/Generic-Routing-Encapsulation-GRE

[14] https://www.slideshare.net/LarryCover/virtualizing-the-network-to-enable-a-software-defined-nfrastructure-sdi?from_action=save

2.2.3. STT

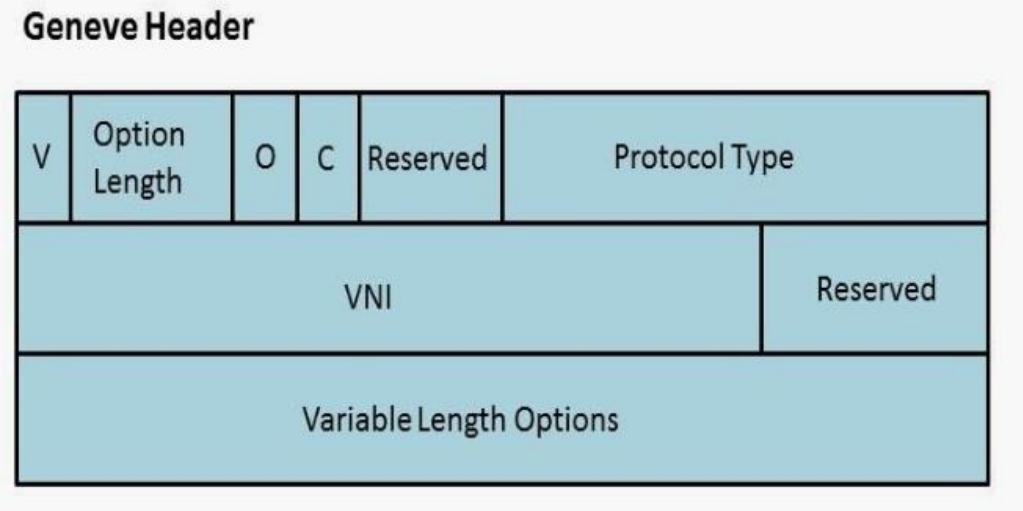
Stateless Transport Tunneling (STT) je dizajniran kako bi poboljšao performanse tako što iskorištava „*TCP segmentation offload*“, TSO u samim mrežnim karticama. Pruža najbolje rezultate u hypervisor-to-hypervisor tunelima jer se iskorištava hardverska akceleracija na mrežnim karticama (NIC).[15] Za razliku od VXLAN-a i NVGRE-a koristi 64 bitni network ID (NVGRE i VXLAN 32 bitni). Ne koristi se značajno u praksi, ali pruža dobru bazu za poboljšanje SDN protokola.

2.2.4. GENEVE

Generic Network Virtualization Encapsulation (GENEVE) je standard koji je nastao kao težnja da se VXLAN, NVGRE i STT povežu i postanu interoperabilni[16]. Autori GENEVE standarda žele definirati samo format enkapsulacijskih podataka. Format podataka mora biti maksimalno fleksibilan kako bi se omogućio budući razvoj i skalabilnost protokola. Trenutni identifikatori u VXLAN-u, NVGRE-u i STT-u su i više nego dovoljni za specificiranje svih potrebnih virtualnih mreža, no budući razvoj bi mogao dovesti do želje da se prenosi više od samih podataka o mreži. Pritom se vodi usporedba primjerice s BGP protokolom koji već vrlo dugi niz godina je neprikosnoven u usmjeravanju zbog skalabilnosti i nadogradivosti.

GENEVE enkapsulirani paketi su dizajnirani za prijenos preko standardne mrežne opreme. Paketi se prenose od jednog „*tunnel endpointa*“ do drugog *unicast* ili *multicast* paketima. Aplikacijama je transparentno i one stvaraju jednake IP pakete kao da komuniciraju preko uobičajene mreže. „*Tunnel endpointi*“ enkapsuliraju IP pakete dodajući im GENEVE zaglavlje s potrebnim podatcima: dužina opcija (ako ih ima) i identifikator tunela. Paket se zatim šalje UDP-om na drugu stranu tunela koji raspakirava zaglavlje, čita opcije (ako ih ima) i šalje paket na krajnju destinaciju.

[15] https://octo.vmware.com/vxlan-stt-and-looking-beyond-the-wire-format/
[16] https://www.redhat.com/en/blog/what-geneve



Slika 10 GENEVE okvir[17]

Slika 10 prikazuje kako izgleda GENEVE okvir.

GENEVE podržava „*Equal-cost multi-path*“ (ECMP), balansiranje prometa preko više putanja kako bi se povećala protočnost, NIC hardware *offload* (koji smo spomenuli kod STT-a), kvalitetu usluge (*QoS*), i izbjegavanje zagušenja. Uključene su sve prednosti VXLAN-a, NVGRE-a i STT-a. Za kontrolni sloj može se koristiti bilo koji protokol, a broj opcija nije ograničen.

2.3. Postojeća SDN rješenja

U SDN svijetu postoji nekoliko najpopularnijih SDN rješenja. To su svakako Cisco ACI i VMware NSX. Radi boljeg razumijevanja i usporedbe opisati ću kako oba SDN rješenja funkcioniraju.

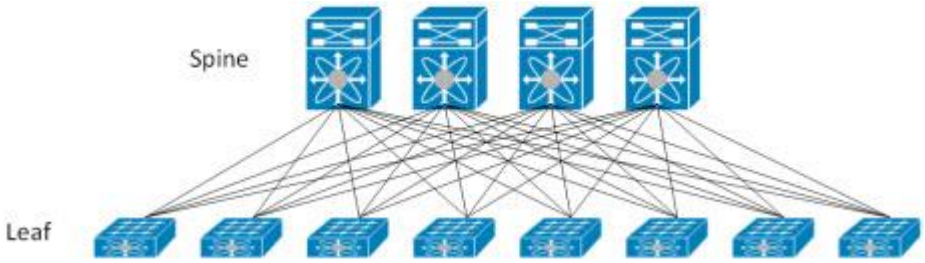
2.3.1. Cisco ACI

Cisco Application Centric Infrastructure je rješenje koje primarno koristi VXLAN kao *overlay* za komunikaciju, ali podržava i NVGRE[18]. Zahtjeva dedicirani Cisco hardware,

[17] https://www.redhat.com/en/blog/what-geneve

[18] https://www.researchgate.net/publication/314082881_Comparison_between_Cisco_ACI_and_VMWARE_NSX

preklopnike iz 9000 serije. Sama fizička infrastruktura je složena u „*leaf-spine*“ arhitekturi, Slika 11:



Slika 11 Leaf spine moderna arhitektura preklopnika[19]

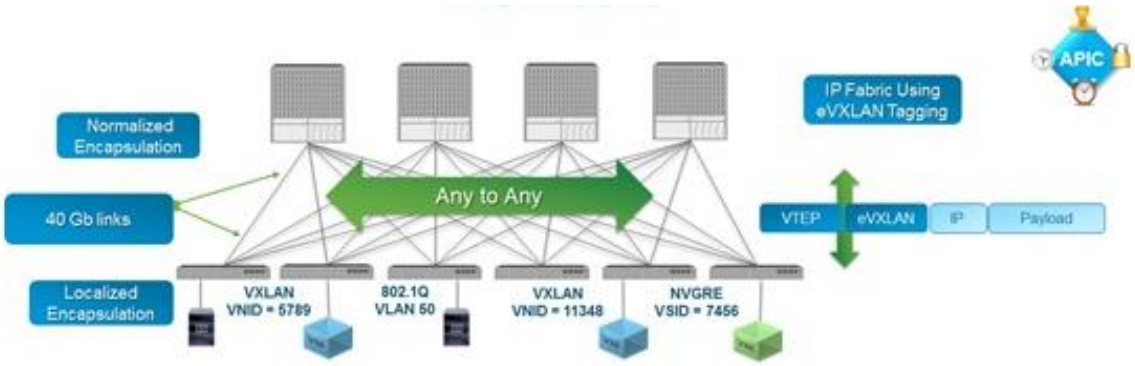
Smisao i prednost arhitekture je da je put paketa predvidljiv, do svake točke se može doći kroz maksimalno 2 skoka. *Leaf* preklopnici služe kao pristupni (*access*, eng.) preklopnici na koje se spajaju uređaji poput poslužitelja, vatrozida, usmjernika. *Spine* sloj preklopnika je sposoban za usmjeravanje prometa i povezuje sve *leaf* preklopnike. *Leaf* i *spine* preklopnici međusobno komuniciraju na trećem sloju OSI modela putem IS-IS dinamičkog usmjerničkog protokola. Tako se omogućava ECMP (*Equal-Cost Multipathing*, engl.) i utilizacija svih linkova, bez *Spanning tree* zatvorenih puteva. *Leaf* preklopnici između sebe zatvaraju VTEP tunele, opisane u poglavlju o VXLAN-u.

Upravljanje Cisco ACI infrastrukturom vrši se putem APIC (*Application Policy Infrastructure Controller*) kontrolera. APIC kontroleri, kojih je potrebno minimalno tri po *fabricu*, su centralni repozitorij za sve politike i konfiguracije.

Cisco ACI ima cilj orijentirati mrežu prema aplikaciji, pa tako uvodi EPG (*EndPoint Group*) kao potpuno novi pristup grupiranju servisa. IP više nije osnovna karakteristika servera. U EPG grupe smještaju se servisi ovisno o njihovoj namjeni i servisi unutar EPG-a mogu međusobno komunicirati neovisno o IP mreži u kojoj se nalaze. Atribut za smještaj servera unutar EPG-a može biti IP adresa, MAC adresa, ime VM-a, karakteristika VM-a (npr. OS) i drugo. Komunikacija između EPG-ova dopušta se zatim putem Ugovora (*Contract*, eng.). Takav pristup potpuno je drugačiji u odnosu na staromodni pristup mreži gdje je osnovna karakteristika servisa IP adresa i na temelju nje se vrše međusobna komunikacija.

Slika 12 prikazuje globalnu Cisco ACI arhitekturu.

[19] https://blog.westmonroepartners.com/a-beginners-guide-to-understanding-the-leaf-spine-network-topology/



Slika 12 Cisco ACI arhitektura[20]

2.3.2. VMware NSX

VMware NSX koristi VXLAN za komunikaciju. NSX ne zahtjeva zaseban hardver, već funkcionira na svim mrežnim uređajima jer se ne naslanja na njihove tehničke karakteristike već koristi uobičajene „*Jumbo frame*“ pakete. NSX-u je bitno da VXLAN paket dođe s jednog NSX uređaja na drugi, bez obzira na to kakva je mreža između. Tu treba reći da se NSX fokusira prvenstveno na virtualna okruženja, bazirana na VMwareovim rješenjima. NSX-u je cilj povezati *hypervisore* i upravljati mrežom interno, bez potrebe za upravljanjem s vanjskim mrežnim uređajima. Unutar NSX-a se tako mogu kreirati različiti NFV (*Network Function Virtualization*) uređaji, poput virtualnog vatrozida, virtualnog usmjernika, virtualnog VPN koncentratora… [21]

2.3.3. Usporedba

Kako Cisco ACI i VMware NSX imaju dosta preklapanja i u velikom dijelu se trude napraviti jednake funkcionalnosti, možemo ih usporediti, Tablica 1:

[20] https://community.cisco.com/t5/data-center-documents/cisco-aci-architecture-simplified/ta-p/3145891
[21] https://www.researchgate.net/publication/314082881_Comparison_between_Cisco_ACI_and_VMWARE_NSX

	Cisco ACI	VMware NSX
Automatizacija mreže	Omogućava punu automatizaciju virtualne i fizičke mreže kroz jednostavan API.	Omogućava automatizaciju virtualnih mreže za virtualne mašine. Ne podržava automatizaciju fizičke mreže.
Spajanje „*legacy*“ sustava	*Legacy* mreže i uređaji mogu biti spoejene direktno na ACI infrastrukturu, uključujući agregaciju linkovi (LACP) i STP protokole.	L2 softverski pristupnici na ESXi infrastrukturi moraju biti korišteni, na koje se mogu spojiti *legacy* uređaji.
Usmjeravanje/*Routing*	Radi se na samim uređajima. Svaki *leaf* je ujedno i *router*.	Odvija se na logičkim softverskim *routerima* instaliranima zasebno.
Dostupnost mreže	Brza konvergencija, ispod sekunde.	Brza konvergencija, moguća sporija konvergencija softverskih uređaja instaliranima kao virtualne mašine.
Umetanje mrežnih usluga	L4 – L7 usluge je moguće umetati kroz Open SDK gdje proizvođači mogu razvijati uređaje koji će komunicirati s APIC-om. Podržava virtualne i fizičke uređaje.	Moguće umetati virtualne i fizičke uređaje kao npr *firewall,* Intrusion *Detection/Preve*ntion, tzv „*Network Introspection*“ Komunikacija kroz API pozive, potrebna dodatna licenca i VM za integraciju.

Vlastita infrastruktura	Zahtjeva isključivo Cisco određene mrežne *switcheve* za implementaciju.	Neovisan o mrežnoj infrastrukturi u pozadini , potrebno je samo omogućiti *routing* i *jumbo frame* okvire.

Tablica 1 Cisco ACI VMware NSX usporedba[22]

[22] https://blog.router-switch.com/wp-content/uploads/2015/09/ACI-vs.NSX_.png

2.4. Network Function Virtualization - NFV

SDN rješenja nastoje biti cjelovita rješenja koja u sebi sadrže osim samog preklapanja i funkcije usmjeravanja prometa, balansiranja i kontrole. Time se poboljšavaju i performanse jer slanjem prometa na uređaje koji su izvan samog SDN-a uvodi se latencija i povećava broj skokova koji promet mora proći da bi došao do krajnjeg uređaja. Jedno od rješenja i NFV, *Network Function Virtualization*, odnosno, virtualizacija mrežnih uređaja. Tako dolazimo do pojma softverskih uređaja i distribuiranih softverskih uređaja. Takvi uređaji mogu biti instalirani vrlo brzo bez potrebe za novom fizičkom opremom. Primjerice, mrežni administratori mogu „podići" *open-source* softverski vatrozid na virtualnoj mašini koja se bazira na x86 platformi.[23] To je vrlo značajan korak u implementaciji *low-cost* mrežne infrastrukture.

Govoreći o performansama NFV uređaja, to je definitivno nešto o čemu treba razmišljati prilikom implementacije. Hardverski uređaji koji imaju dedicirane procesore za specifične funkcije mogu garantirati određene performanse, dok se kod NFV uređaja treba voditi preporukama za instalaciju *vendora* i provesti testove prije puštanja u produkciju takvih uređaja.

Distribuirani NFV uređaji dovode još jednu praktičnost ovakvog pristupa. Primjerice, virtualni klasterirani *firewall* možemo imati instaliran na svakom *hypervisoru* i time smanjiti latenciju koju bi promet imao da mora proći uvijek kroz isti fizički *firewall* u mreži. Još jedan benefit je automatizacija ovakvog pristupa. *Firewall* se može automatizirano podići čim se instaliran novi *hypervisor* s virtualnim mašinama. Takvim NFV uređajima se upravlja centralizirano s jednog mjesta čime se i pojednostavljuje sama administracija uređaja. Primjer jednog od takvih uređaja kojeg ćemo pokazati u praksi je Open vSwitch.

2.4.1. Open vSwitch

Open vSwitch je osmišljen kako bi se unaprijedilo preklapanje prometa između virtualnih mašina i vanjskog svijeta. Na linux *hypervisorima* najčešće se koristi ugrađeni L2 preklopnih koji jest brz i pouzdan. No kada dođemo do multi-server implementacija u okruženjima sa dinamičkim klijentima (*endpoint,* eng.) i potrebe za preusmjeravanjem prometa ovisno o

[23] http://www.ttcenter.ir/ArticleFiles/ENARTICLE/3431.pdf

logičkim slučajevima, dolazimo do potrebe za naprednijim uređajem od ugrađenog *switcha*.[24]

Open vSwitch podržava sve navedene slučajeve. Brza migracija *hostova* između *hypervisora* primjerice, omogućava ne samo zadržavanje postojeće konfiguracije pristupnih lista (*access list*, eng.) i pravila o kvaliteti usluge, već omogućava i zadržavanje mrežne sesije. Open vSwitch nije samo L2 uređaj, već je sposobna sagledavati upravljati prometom na višim mrežnim OSI razinama (L2-L7). Open vSwitch podržava bazu mrežnih stanja (OVSDB) kako bi omogućio okidanje određenih promjena u slučaju detekcije promjene na mreži. To može iskoristiti, primjerice, softver za orkestraciju kako bi pratio i poduzeo određene akcije na migracije virtualnih mašina. Open vSwitch podržava OpenFlow protokol.

OpenFlow protokol je standardizirani protokol za komunikaciju sa switchevima neovisno o vendoru. To omogućava kontrolu ponašanja switcheva u cijeloj mreži na dinamički način s jednog centraliziranog mjesta (Openflow kontrolera) Openflow kontroler je kontrolni sloj (*control plane,* eng.) i daje *switchu* uputu što napraviti s paketima koji dolaze[25]. Prednost ovakvog pristupa je centralizacija kontrolnog od podatkovnog sloja što znači konzistentno ponašanje i konfiguraciju kroz cijelu mrežu. Nadalje, OpenFlow pruža brojne dodatne mogućnosti upravljanja L2 prometom, ali to je cijela zasebna tema i neće biti detaljno obrađena u ovom radu. Open vSwitch podržava rad i u standardnom načinu i u OpenFlow načinu. Ključne komponente OVS-a:

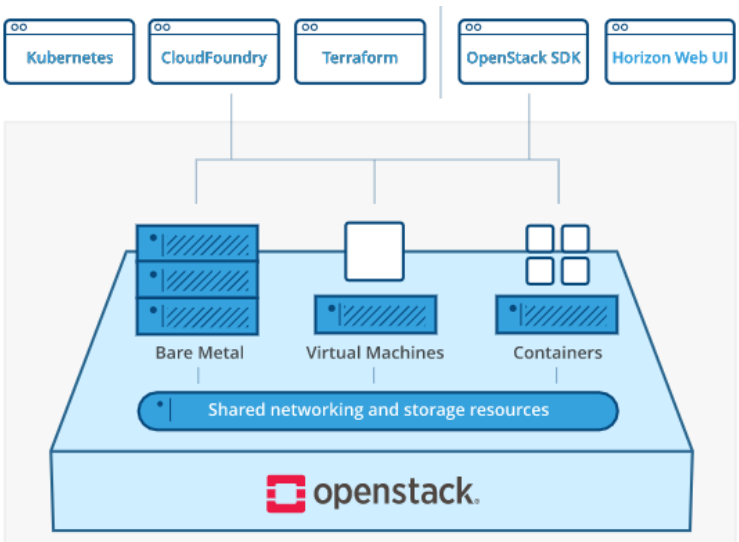
- VSWITCHD – core komponenta za komunikaciju s host OS-om
- OVSDB-SERVER – sadrži konfiguraciju switcha
- Kernel Module – za primjenu učitanih (*cached,* eng.) *fast-path* pravila

[24] http://docs.openvswitch.org/en/latest/intro/what-is-ovs/
[25] https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/

3. OpenStack

OpenStack je nastao 2010. godine kada je Rackspace[26], poznati cloud *provider* odlučio ponovno napisati kod za svoje Cloud servere i učiniti ga otvorenim (*open source*, eng.). U isto vrijeme NASA-ina tvrtka koja je radila software, Anso Labs, napravila je beta kod za Nova – cloud kontroler. Tvrtke su prepoznale dobrobiti međusobne suradnje, i odlučile ujediniti suradnju u projekt koji se nazvao OpenStack s misijom „Stvoriti višenamjensku *open source* oblačni računarsku platformu koja će zadovoljiti potrebe privatnih i javnih cloudova bez obzira na veličinu, uz jednostavnost implementacije i skalabilnost“.[27]

OpenStack je cloud operativni sustav koji kontrolira skupine računalnih, spremišnih i mrežnih resursa u podatkovnom centru, svima se upravlja kroz centralnu konzolu koja administratoru pruža uvod u stanje kompletnog sustava, a korisnicima jednostavno upravljanje vlastitim resursima, Slika 13. OpenStack pruža programabilnu infrastrukturu sa setom API sučelja koja pruža jedinstvenu platformu za virtualne mašine, kontejnere i fizičke mašine. Bitno je napomenuti da je OpenStack *open source* platforma što znači da je otvorenog koda, svatko može pristupu kodu i nadograditi kod i „skinuti“ besplatno sustav za korištenje (naplaćuje se samo partner podrška u slučaju potrebe i/ili „*hostanje*“ resursa).[28]



Slika 13 OpenStack[28]

[26] https://www.rackspace.com/
[27] http://OpenStackceeday.com/blog/20180427-6-milestones-history-OpenStack
[28] https://www.OpenStack.org/software/

OpenStack omogućuje korisnicima *deploy* virtualnih mašina i resursa. Pojednostavljuje skaliranje na način da se prateći iskoristivost resursa može na jednostavan način dodati ili oduzeti resursi kako bi sustav bio optimalno iskorišten.

Što je uopće *cloud*? *Cloud* računarstvo se bazira na pružanju računalnih resursa krajnjim korisnicima udaljenim pristupom na siguran i pouzdan način. Krajnjim korisnicima je relativno nebitno što se krije „ispod haube“ *clouda*, već im je bitno da resursi koje plaćaju pouzdano rade.

3.1. OpenStack komponente

OpenStack ima vrlo modularan dizajn. Radi toga sastoji se od mnogo komponenata. Da bismo bolje razumjeli koncept cijelog sustava objasnit ću pobliže osnovne komponente OpenStacka. Svaka komponenta upravlja zasebnim resursom koji može biti virtualiziran za potrebe korisnika. Upravo odvajajući svaki virtualizirajući resurs u zasebnu komponentu čini OpenStack toliko modularnim. Ako pojedini usluga resursa nije potrebna u korištenju OpenStacka, to u pravilu znači da je opcionalna i da ne treba biti instalirana u našu implementaciju OpenStacka.

Komponente su logički podijeljene u tri skupine:

1. Kontrolna (*Control*)
2. Mrežna (*Network*)
3. Računalna (*Compute*)

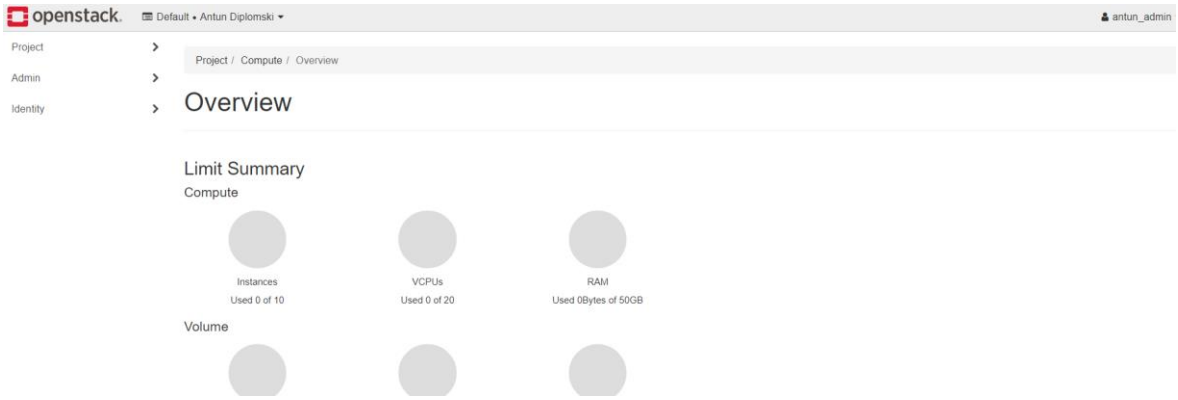
Kontrolna skupina pokreće API (*Application Programming Interface*) sučelje, web sučelje, bazu podataka i upravljanje porukama. Mrežna skupina pokreće mrežne servisne agente, a računalna komponenta su *hypervisori* zaduženi za rad virtualnih mašina. Sve skupine koriste bazu podataka i poruke koji se u manjim instalacija nalaze na kontrolnom čvoru (*node*, eng.).

Horizon je *web* bazirano sučelje koje služi kao kontrolna ploča (*Dashboard*, eng.). Kroz Horizon možemo upravljati svim komponentama OpenStacka. Horizon je, također, i centralno sučelje za krajnje korisnike kako bi upravljali svojim cloud uslugama. Sljedeće su karakteristike Horizona[29]:

[29] https://docs.OpenStack.org/horizon/latest/contributor/intro.html#contributor-intro

- Pruža „*out-of-the-box*“ podršku za sve OpenStack komponente
- Nadogradiv je, bilo tko može dodati novu komponentu
- Upravljiv, jednostavan za upotrebu i za navigiranje
- Konzistentan, vizualizacije i interakcije su konzistentne kroz cijelo sučelje
- Stabilan, ima pouzdan API s naglaskom na kompatibilnost
- Uporabljiv, sučelje primamljivo oku korisnika

Horizon u zadanoj instalacija ima tri prozora kontrolne ploče: *Project, Admin, Identity.*



Slika 14 Horizon Dashboard

Keystone je autentifikacijski i identifikacijski servis koji služi i za upravljanje korisničkim računima i rolama u OpenStack okruženju. To je ključni servis kojega koriste sve ostale OpenStack komponente i prvi je servis koji treba biti instaliran u okruženju. Keystone vrši autentifikaciju korisnika i sustava šaljući validirani autentifikacijski token između svih OpenStack servisa, primjerice između podatkovnog (*Storage*) i računarskog (*Compute*). Radi toga ovaj servis mora biti prvi instaliran i pomoću njega moraju biti kreirane role, korisnici i *tenanti* za cloud infrastrukturu. *Tenant* je projekt koji se sastoji od resursa poput *imagea*, korisnika, instanci i mreža koji pripadaju samo tom projektu. I stvarnome svijetu *tenant* može biti svaki odjel unutar tvrtke koja se koristi OpenStackom kao *Cloud* sustavom ili svaka tvrtka ukoliko je OpenStack javni *cloud provider*. Korisnik može pripadati jednom ili više *tenanta*. Korisnici unutar *tenanta* mogu imati različite role, primjerice „administrator“ ili „*read-only*“ ili „*normal*“. [30]

Glance je servis koji služi za registraciju, otkrivanje i korištenje slika (*image*, eng.) za virtualne mašine koje se koriste u OpenStack okruženju. Slike koje su dostupne kroz Glance

[30] https://docs.openstack.org/keystone/latest/getting-started/architecture.html

mogu biti pohranjene na različitim stvarnim lokacijama, od npr. lokalnog datotečnog sustava sve do OpenStack sustava za pohranu objekata, *OpenStack Object Storage*. Osim slika operativnih sustava, Glance može pohranjivati još i definicije metapodataka za primjenu kod pokretanja slika OS-ova. To je jednostavan katalog ključeva i vrijednosti.[31]

Nova je računalna komponenta (*compute*, eng.) OpenStacka. Nova je komponenta na kojoj se izvršava virtualizacija i koja omogućava pokretanje više instanci operativnih sustava u isto vrijeme omogućavajući kreiranje visoko skalabilnog i redundantnog Cloud okruženja. Nova podržava više vrsta virtualizacije, uključujući QEMU, KVM, VMware.[32]

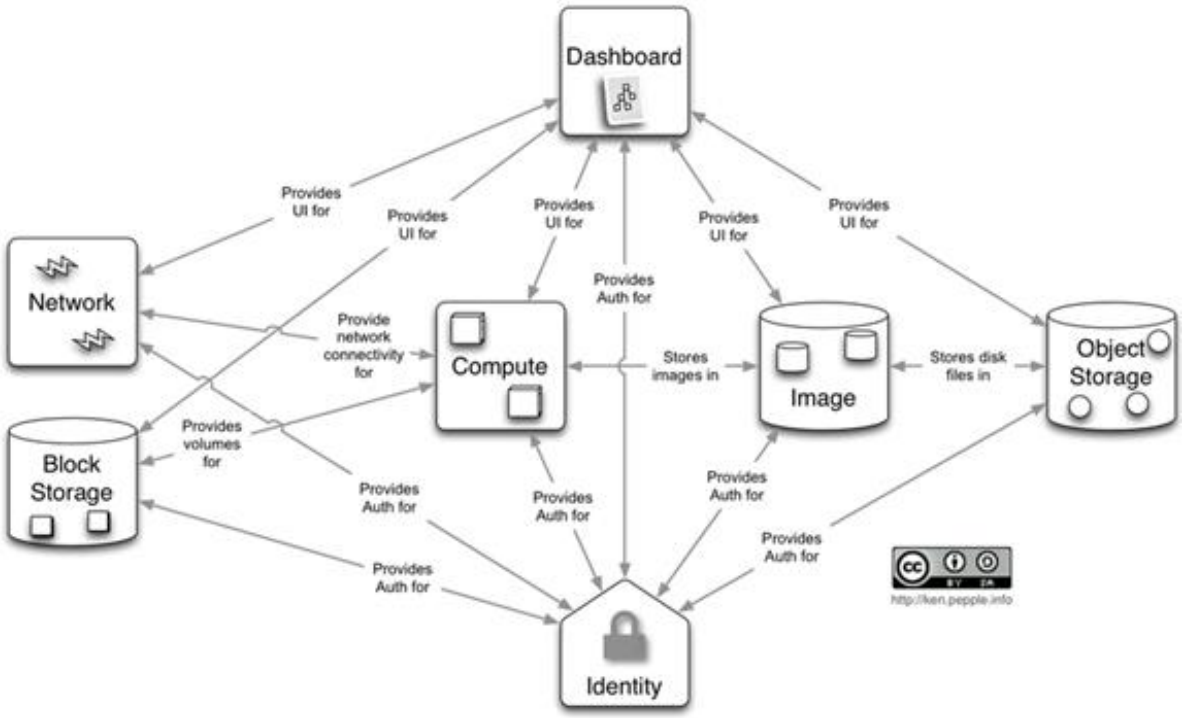
Swift je *OpenStack Object Storage* komponenta, servis koji omogućava kreiranje vrlo skalabilnoga redundantnog *storagea* na raznom hardveru. Sa Swiftom možemo pohraniti gotovo neograničen broj objekata u našu OpenStack okolinu, ograničeni smo samo hardverskim resursima. Redundantna osobina Swifta je idealna za arhiviranje i dugotrajnu pohranu podataka (poput log zapisa), i također za razne image virtualnih mašina.[33]

Cinder je *OpenStack Block Storage* komponenta koja upravlja volumenima koj isu priključeni na *compute* instance. Volumeni su diskovi koje koriste virtualne mašine, a najbolja analogija je priključene USB podatkovnog ključa u računalo. Svaka instanca može bit pridružena samo jednoj *compute* instanci – virtualnoj mašini. Volumeni se mogu predstaviti kroz, primjerice, iSCSI protokol.[34]

Napokon dolazimo i do nama najbitnije komponente u ovome radu, Neutron komponente. Neutron je SDN komponenta OpenStacka zaduženo za sve mrežne operacije unutar OpenStack okruženja. Sa SDN-om, možemo opisati kompleksne mreže u sigurnom *multitenant* okruženju koje prevazilaze probleme flat mreža i VLAN-ova. SDN također omogućava jednostavno „uključivanje“ raznih NFV uređaja poput firewalla, load balancer i sl.[35]

Na Slici 15 možemo vidjeti međuovisnost ovih ključnih OpenStack komponenti.

[31] https://docs.OpenStack.org/glance/latest/
[32] https://docs.openstack.org/nova/latest/
[33] https://docs.openstack.org/swift/latest/
[34] https://wiki.openstack.org/wiki/Cinder
[35] https://docs.openstack.org/neutron/pike/install/overview.html



Slika 15 Međuovisnost OpenStack komponenti[36]

Fizička infrastruktura OpenStack instalacije će se najčešće sastojati od čvorova podijeljenih u sljedeće kategorije:

1. Kontrolni čvor/*Controller node*
    - Pokreće API servise za sve OpenStack komponente
    - Pokreće bazu podataka i sustave poruka
    - Pokreće Horizon kontrolnu ploču za upravljanje OpenStackom
2. Mrežni čvor/*Network node*
    - Pokreće DHCP servis i *metadata* servise za mrežne usluge
    - Pokreće virtualne usmjernike
    - U manjoj infrastrukturi može biti spojen s kontrolnim čvorom
3. Računalni čvor/*Compute node*
    - Pokreće hypervisor poput KVM-a, HyperV-a Xen-a ili containere tipa LXC
    - U slučajevima korištenja distribuiranog usmjernika (DVR – *Distributed Virtual Router*) ovaj čvor pokreće svoju instancu DVR-a
4. Podatkovni čvor/*Storage node*

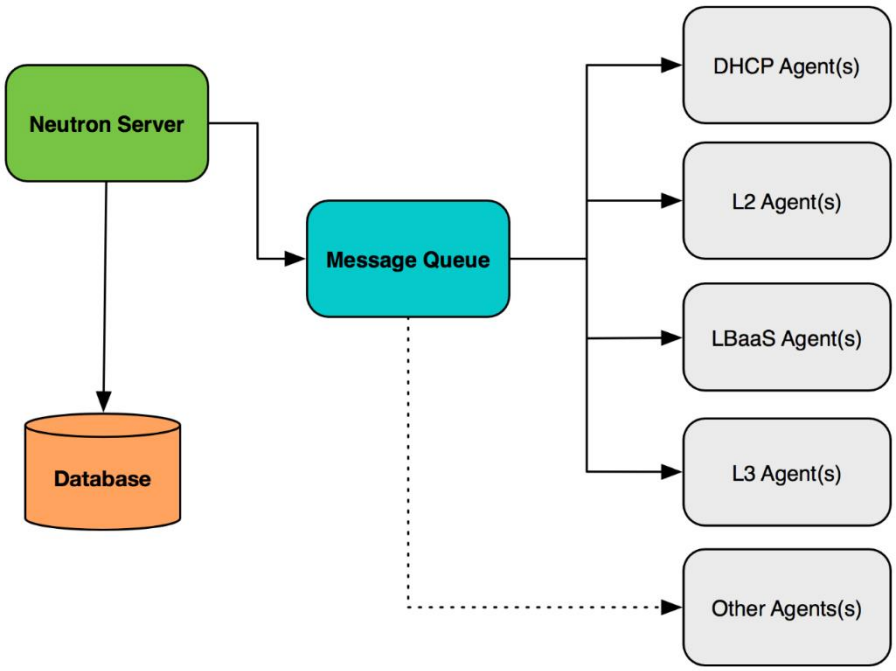
[36] https://cloudify.co/2014/07/18/OpenStack-wiki-open-cloud.html

- Pokreće softvere vezane uz Storage, primjerice Cinder ili Swift
- U pravilu ne pokreću bilo kakve *networking* servise ili agente

3.2. OpenStack mreža – Neutron

Neutron, OpenStack *Networking* servis nadopunjuje ostale *core* servise, *Nova, Glance, Keystone, Cinder, Swift* i *Horizon* kako bi se pružilo kompletno cloud rješenje. Neutron predstavlja i API sučelje za korisnike i prosljeđuje zahtjeve prema *networking* servisima. Korisnici mogu definirati i konfigurirati mrežnu povezivost u *cloudu*, a administratori imaju mogućnosti koristiti razne tehnologije dostupne na tržištu kako bi ostvarili *cloud* sustav.

OpenStack servisi mogu biti raspršeni između različitih servera kako bi se omogućila otpornost i redundantnost, ili se mogu instalirati na jedan čvor (engl. *node*). Kao i ostali servisi, Neutron zahtjeva komunikaciju s bazom podataka kako bi pohranio mrežnu konfiguraciju. Na Slici 16 možemo vidjeti pojednostavljenu arhitekturu Neutron komunikacije:



Slika 16 Neutron arhitektura[37]

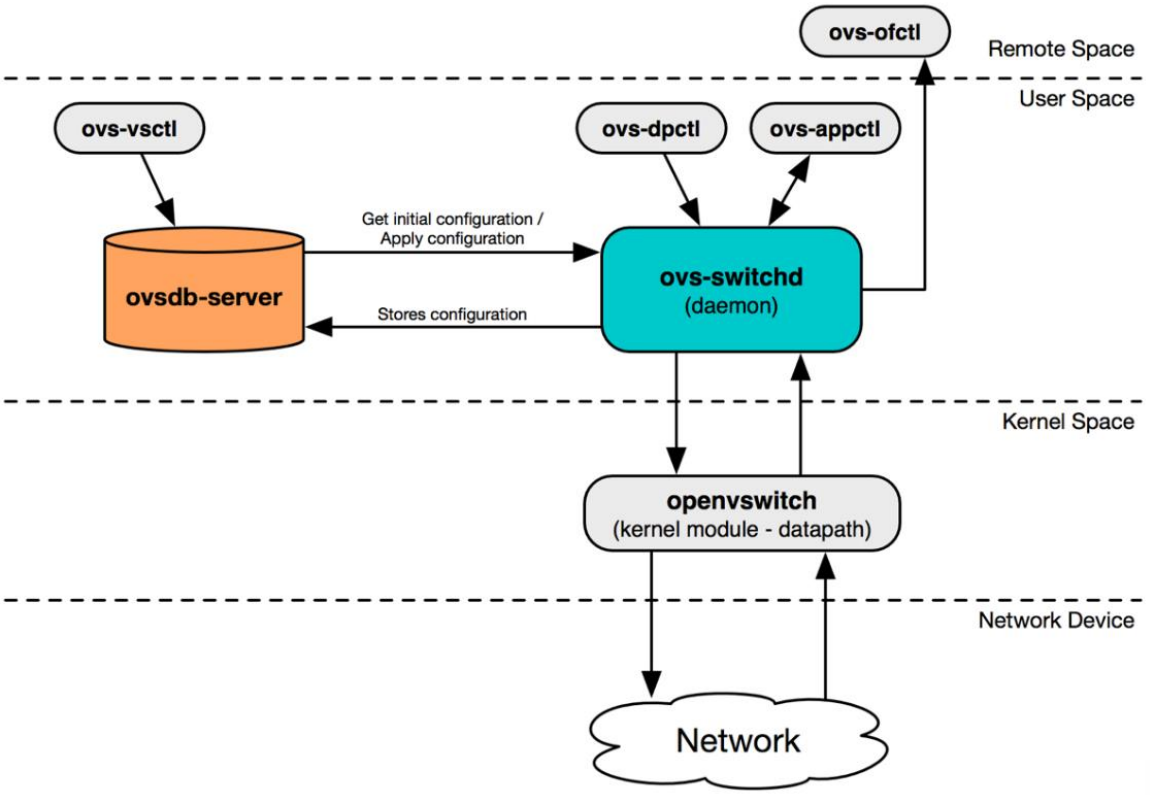
[37] James Denton: Learning OpenStack Networking, 2018.

Na slici je prikazano kako se Neutron Server spaja na bazu podataka gdje je smještena mrežna konfiguracija. Neutron prima API zahtjeve od korisnika i servisa i komunicira s agentima putem „Reda čekanja poruka“ (*Message Queue*, eng.). U tipičnoj implementaciji, mrežni agenti su raspršeni preko kontrolnih i *compute* čvorova i obavljaju zadaće za zadani čvor.

3.2.1. Mogućnosti Neutrona

Preklapanje (*Switching*, engl.) je prva i osnovna mogućnost Neutrona. Virtualni *switch* definira se kao softverska aplikacija ili servis i spaja virtualne mašine na virtualne mrežne na podatkovnom sloju OSI modela, L2. Neutron podržava više virtualnih *switching* platformi, uključujući standardne Linux OS *switcheve* i Open vSwitch. O Open vSwitchu smo pisali ranije u ovome radu pa ga nećemo ponovno pobliže objašnjavati. Nama je bitno da podržava *overlay* mrežne protokole poput NVGRE-a i VXLAN-a. Možemo još dodatno ponoviti da Open vSwitch u OpenStacku ima sljedeće komponente, Slika 17:

- Kernel modul – Ekvivalent ASIC čipa na hardverskom *switchu*. *Data plane* uređaja koji procesuira sve pakete
- *vSwitch* daemon – Linux proces koji je pokrenut u OS-u na svakom fizičkom *hostu* i upravlja *kernel* modulom
- *Database* server – OVSDB – lokalna baza na svakom fizičkom *hostu* koja sadrži konfiguraciju virtualnih *switcheva*



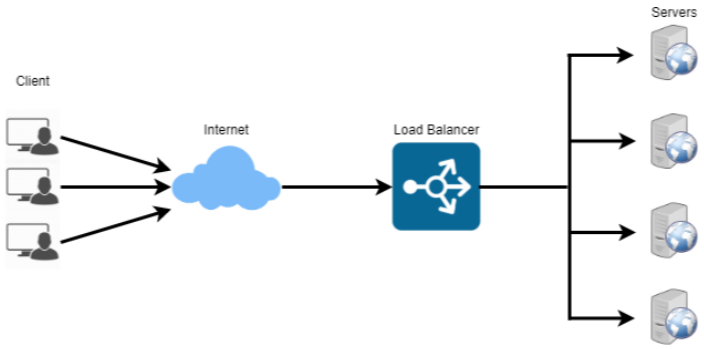
Slika 17 Dijagram Open vSwitch komponenti[38]

Virtualni *switchevi* mogu biti korišteni i za ostvarivanje komunikacije između instanci, a moguće je ostvariti i kontroliranje fizičkih uređaja izvan OpenStack okoline, poput hardverskih *switcheva*, no to je izvan opsega ovog rada.

Usmjeravanje (*Routing*, engl.) je još jedna od mogućnosti koje omogućava Neutron. *Routing* i NAT (*Network Address Translation*, engl.) mogući su kroz korištenje standardnih Linux funkcija – *IP forwarding*, *iptables* i *network namespaces*. Svaki *network namespace* sastoji se od zasebne *routing* tablice, sučelja (*interface*, eng.) i *iptables* procesa koji omogućava filtriranje prometa i NAT funkcionalnosti. NAT je prevođenje IP adresa. Postoji *Destination* NAT gdje mijenjamo odredišnu IP adresu u neku drugu (zamjena *destination* IP polja u IP paketu) ili *Source* NAT gdje mijenjamo polazišnu IP adresu (Zamjena *source* IP polja u IP paketu). Klasičan slučaj korištenja NAT-a je prilikom pristupa na Internet iz privatne uredske mreže gdje se *source* privatna IP adresa mijenja najčešće javnom adresom našeg internet usmjernika. Upravo korištenje *namespacea* omogućava razdvajanje mreža tako da ne moramo brinuti o mrežnim preklapanjima između *tenanta* koje koriste korisnici.

[38] James Denton: Learning OpenStack Networking, 2018., Fig 5.1

Balansiranje prometa (*Load Balancing*, eng.) je funkcija Neutrona koja nije postojala od samih početaka. Nudi mogućnosti balansiranje mrežnog prometa što je izuzetno korisno u npr. slučajevima skaliranja prometa, tipično za web servere koji imaju veliki broj zahtjeva u određeno doba. *Load balancing* nudi mogućnost distribuiranja klijentskih zahtjeva na više instanci ili servera. Korisnik može kreirati monitore, limite konekcija, ili profile perzistencije za promet koji prolazi preko load balancera. Neutron pruža mogućnosti *plugina* koji utilizira HAProxy, vrlo popularnu inačicu *open source load balancera*. Tipična shema load balancera na Slici 18:



Slika 18 Load balancer[39]

OpenStack pruža i mogućnosti vatrozida, i to kroz dva pristupa: Sigurnosne grupe (*Security group*, engl.) i *Firewall as a Service (FWaaS)*. Security grupe se baziraju na grupiranje resursa u grupe koje dijele slične funkcionalnosti i međusobno smiju komunicirati. U pozadini, *Security* grupe funkcioniraju na razini porta instance koristeći „*iptables*“ servise ili OpenFlow. FWaaS funkcioniraju također na razini porta instanci, ali mogu biti primijenjene i na portove, primjerice, usmjernika, od verzije FWaaS v2. *Firewall* se sastoji od dvije osnovne politike (*policy*, engl.), ulazni *policy* i izlazni *policy*. Svaki od tih *policya* se sastoje od skupine pravila, a pravila se sastoje od skupine atributa poput ishodišne i odredišne IP adrese, protokola ili portova. Pravilo dopušta (*allow*, engl.) ili zabranjuje (*deny*, engl.) promet koji je zahvaćen atributima.[40]

VPN (*Virtual Private Networks*, engl.) mogućnost omogućava zaštićeno proširenje privatnih mreža na druge lokacije preko, npr. Interneta ili zakupljenih ISP linija. Neutron pruža set

[39] https://www.educative.io/collection/page/5668639101419520/5649050225344512/5747976207073280
[40] https://docs.OpenStack.org/neutron/pike/admin/fwaas.html

API instrukcija koje omogućavaju korisnicima kreiranje IPSec VPN tunela od Neutron usmjernika do udaljenih pristupnika.

3.2.2. Tipovi mreža u Neutronu

OpenStack Networking podržava resurse odvojene po *tenantu*/projektu što uključuje i mrežne resurse. Svaki *tenant* može imati vlastite mreže i *routere* neovisno o drugim *tenantima*. Korisnici unutar *tenanta* mogu zadavati razne mreže neovisno o tome postoje li te mreže u nekom drugom *tenantu*. Administrator također može i ograničiti adresne prostore koje *tenanti* mogu koristiti.

Dva su tipa mreža u OpenStacku:

1. *Tenant*/*project network* – virtualna mreža kreirana unutar *tenanta* za *tenant*. Fizički detalji te mreže nisu potrebni niti poznati *tenantu*.
2. *Provider network* – Virtualna mreža kreirana kako bi se mapirala na fizičku mrežu. Direktno se veže na fizičku mrežu/VLAN izvan OpenStacka i omogućava komunikaciju s vanjskim svijetom.

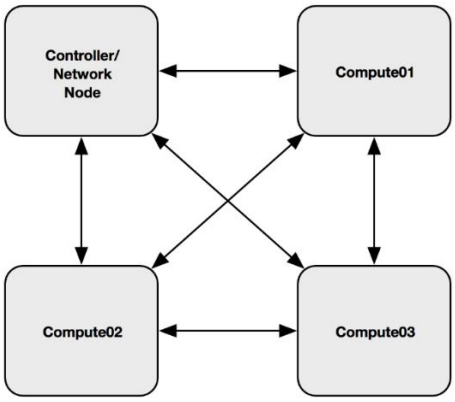
Project network pruža mogućnosti povezivanja resursa unutar *tenanta*/projekta. Korisnici mogu kreirati, modificirati i brisati *project* mreže. Svaka *project* mreža je izolirana od drugih *project* mreža VLAN-om ili „*segmentation* ID-em“ (VXLAN). *Provider* mreža pruža stalnu povezivost mrežama izvan *clouda* i tipično ju kreira *cloud* administrator. Osnovna razlika između ovih mreža se može vidjeti i tijekom procesa podizanja mreža. *Provider* mreže kreiraju administratori za pojedini projekt i mogu biti dodijeljene pojedinom projektu ili dijeljene između projekata. Project mreže se kreiraju unutar projekta kako bi ih koristili servisi i virtualne mašine unutar *tenanta* i ne mogu biti dijeljene između projekata/*tenanta*. Kada se kreira *provider* mreža, administrator može odrediti specifične detalje koji nisu dostupni običnim korisnicima, poput tipa mreže, fizičkog sučelja koji će se koristiti, identifikatora segmentacije, primjerice VLAN ID ili VXLAN VNI. Project mreže mogu imati iste ove atribute, ali korisnik ih ne može određivati, već su automatski dodijeljeni na temelju zadanih vrijednosti Neutrona.

U daljnjem tekstu biti će govora o sljedećim osnovnim mrežnim resursima unutar OpenStacka:

Resurs	**Opis**
Subnet	Blok IP adresa kojim se alociraju portovi na mreži.
Port	Točka spajanja za priključivanje pojedinog uređaja, poput virtualne mrežne kartice (vNIC) virtualne mašine. Atributi porta su MAC adresa i fiksirana IP adresa *subneta.*
Router	Virtualni uređaj koji pruža usmjeravanje između *self-service* mreža i *provider* mreža.
Security group	Set pravila *firewalla* koja kontroliraju ulazni i izlazni promet.
DHCP	Agent koji se bavi dodjelom IP adresa za servise i virtualne mašine.
Metadata	Metapodatci koji služe za dodatne podatke virtualnih mašina prilikom podizanja.

Tablica 2 Osnovni mrežni resursi u OpenStacku

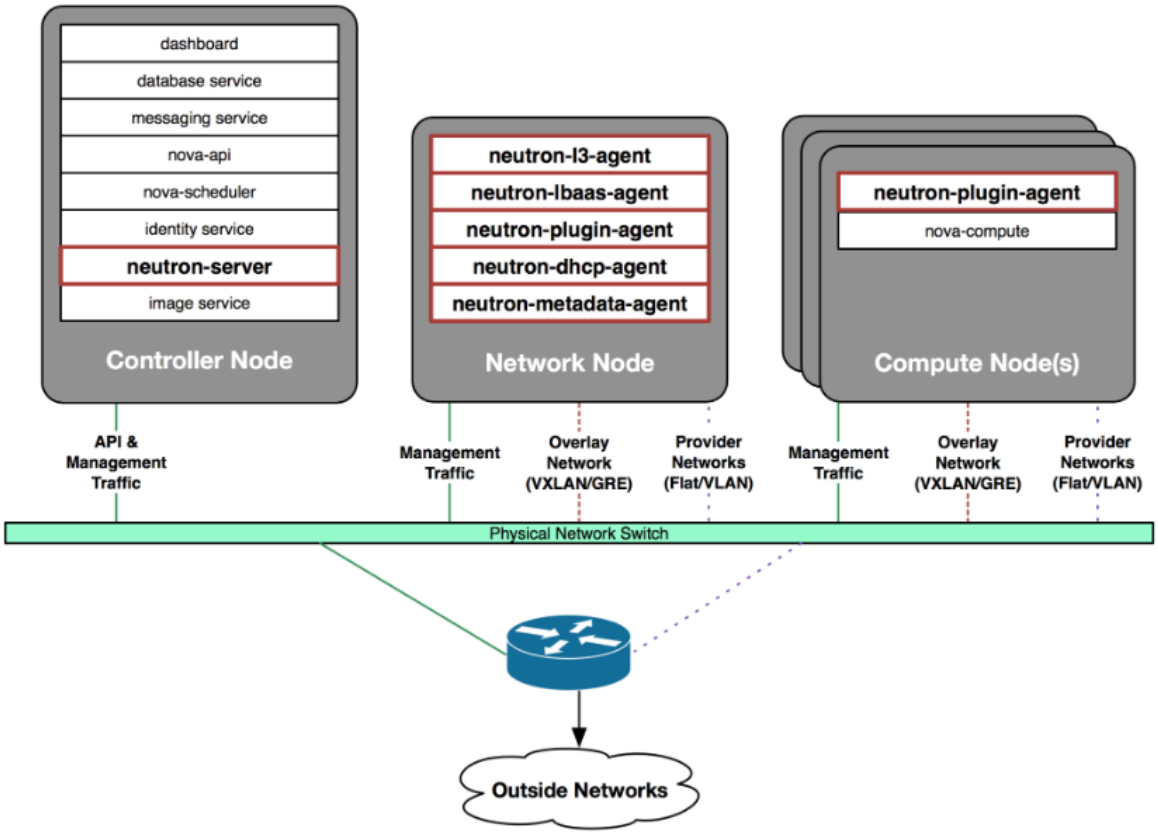
Što se tiče *overlay* mreža, OpenStack podržava VXLAN, NVGRE i GENEVE. Te tehnologije su već pobliže objašnjene u prijašnjem dijelu rada. Neutron zadano koristi VXLAN kao *overlay* mrežnu tehnologiju. Stvaraju se *Point-to-point* tuneli između svih *network* i *compute* čvorova stvarajući tzv. *Mesh* mrežu gdje je svaki *host* spojen sa svakim *hostom*. Uzmimo primjer clouda koji se sastoji od jednog kontroler/network čvora i tri *compute* čvora. *Mesh overlay* VXLAN povezivost bi izgledala kao na Slici 19:



Slika 19 VXLAN mesh[41]

Promet između instanci na bilo kojem *hostu* bi putovao između *Layer 3 endpointa* bez obzira na *Layer 2* mrežu u pozadini.

U OpenStack instalaciji, razlaganje Neutron servisa između *hostova* izgleda kao na Slici 20:



Slika 20 Neutron komponente[42]

[41] James Denton: Learning OpenStack Networking, 2018., Fig 1.2
[42] James Denton: Learning OpenStack Networking, 2018., Fig 1.3

Neutron Server je instaliran na *Controller* čvoru, s Neutron agentima zaduženima za implementaciju pojedinog mrežnog resursa instaliranima na Network čvoru. Svaki *Compute* čvor pokreće network dodatak (*plugin*, engl.) zadužen za mrežne operacije na tom *hostu*.

U OpenStack mrežnoj instalaciji možemo razlikovati četiri tipa mrežnog prometa.

1. *Management*
2. API
3. *External*
4. *Guest*

Svaki od ovih tipova mrežnog prometa može imati zasebno fizičko sučelje, ili mogu biti grupirani po sučeljima (čak i svi u jedno) i logički podijeljeni, npr. u VLAN-ove.

Management mreža je interna mrežna koja služi za upravljanje i internu komunikaciju između *hostova* i servisa, poput servisa za poruke i baza podataka. Smatra se *control plane* mrežom. Svi *hostovi* međusobno komuniciraju putem ove mreže. Transferi *imagea* se također mogu odvijati preko ove mreže. *Management* mreža je izolirana i najčešće nema izlaza prema vanjskom svijetu.

API mreža služi za upućivanje API poziva unutar clouda i smatra se *control plane* mrežom. Izolirana je, nema izlaz prema vanjskom svijetu.

Eksterna mreža (*External network,* engl.) je *provider* mreža na kojoj se nalaze Neutron usmjernici koji omogućuju pristup vanjskim mrežama. Kada usmjernik konfiguriramo i priključimo na eksternu mrežu, ta mreža postane izvor plutajućih (*floating,* engl.) IP adresa za servise i virtualne mašine priključene na usmjernik. IP adrese u eksternoj mreži su „rutabilne“, što znači da se njima može pristupiti izvana i da one imaju pristup vanjskom svijetu i internetu. Smatraju se *data plane* prometom. Neutron tagira ove mreže VLAN ID-em slijedeći postavke administratora.

Gostujuća mreža (*Guest network*, engl.) je mreža koju koriste servisi i virtualne mašine. To može biti lokalna mreža unutar *tenanta*, *flat* mreža, VLAN mreža ili VXLAN/NVGRE/GENEVE mreža. Ove mreže pružaju mrežnu povezivost virtualnim mašinama i smatraju se data plane mrežama.

Važno je razlikovati i sljedeće mrežne pojmove u OpenStack okruženju:

- Mreža/*Network* – Izolirana L2 *broadcast* domena. Tipično mreže su rezervirane za *tenante* koji ih kreiraju, ali u nekim slučajevima mogu biti i dijeljene. Mreža je jezgreni entitet Neutrona API-a. *Subneti* i *portovi* moraju uvijek biti pridruženi mreži
- *Subnet* – blok adresnog prostora iz kojega IP adrese mogu biti dodijeljene virtualnim mašinama. Svaki *subnet* mora biti dodijeljen mreži. Više *subneta* može biti dodjeljeno jednoj mreži i ne moraju biti uzastopni.
- *Port* – U OpenStacku, port predstavlja virtualni port na *switchu*. Sučelja virtualnih mašina su mapirana/spojena na Neutron portove, a portove definira MAC adresa IP adresa sučelja na koje su spojeni. Definicije Neutron portova su pohranjene u Neutron bazi podataka.

3.2.3. ML2 plugin

U OpenStack *Networkingu*, priključak (*plugin,* engl.) je osnovna komponenta koja omogućuje funkcionalnosti mreže. Postoje *„core“* i „*service“* priključci. *Core* priključak omogućuje osnovne Neutron funkcionalnosti i zadužen je za osnovne logičke mrežne komponente – mreže, portove i *subnete*. *Service plugin* su dodatni mrežni servisi poput usmjeravanja, *load balancinga* i vatrozida.

Osnovni mrežni priključak za Neutron je „*Modular Layer 2 Plugin*“ (ML2). Prije ML2, Neutron je koristio jednostavni ugrađeni priključak koji je bio ograničen mogućnostima i nije imao mogućnosti ekstenzije i skalabilnosti. ML2 *plugin* omogućuje ekstenzije i podržava heterogene mrežne arhitekture koje mogu koristiti više tehnologija odjednom. Zamijenio je „*Linux bridge core plugin*“ i “*Open vSwitch core plugin“*. “*Open vSwitch core plugin“* nije isto što i Open vSwitch i Open vSwitch agent.[43]

Dva su tipa drivera koje koristi ML2 plugin. „*Typedriver*“ definira tipove mreža koje implementiramo. Održava stanje mrežno ovisno o tipu, validira mrežne atribute, i opisuje mrežne segmente koristeći labele, *segmentation ID* ili tip mreže. Podržani tipovi mreža su: *local, VLAN, flat, VXLAN, NVGRE, GENEVE.* Lokalna mreža je izolirana od ostalih mreža i čvorova. Virtualne mašine koje su spojene na lokalnu mrežu mogu komunicirati s drugim instancama na istoj mreži i na istom *compute* čvoru, ali ne mogu komunicirati s instancama na drugim čvorovima neovisno jesu li na istoj mreži. Ovo je vrlo ograničen tip mreže koji se

[43] https://docs.OpenStack.org/neutron/pike/admin/config-ml2.html

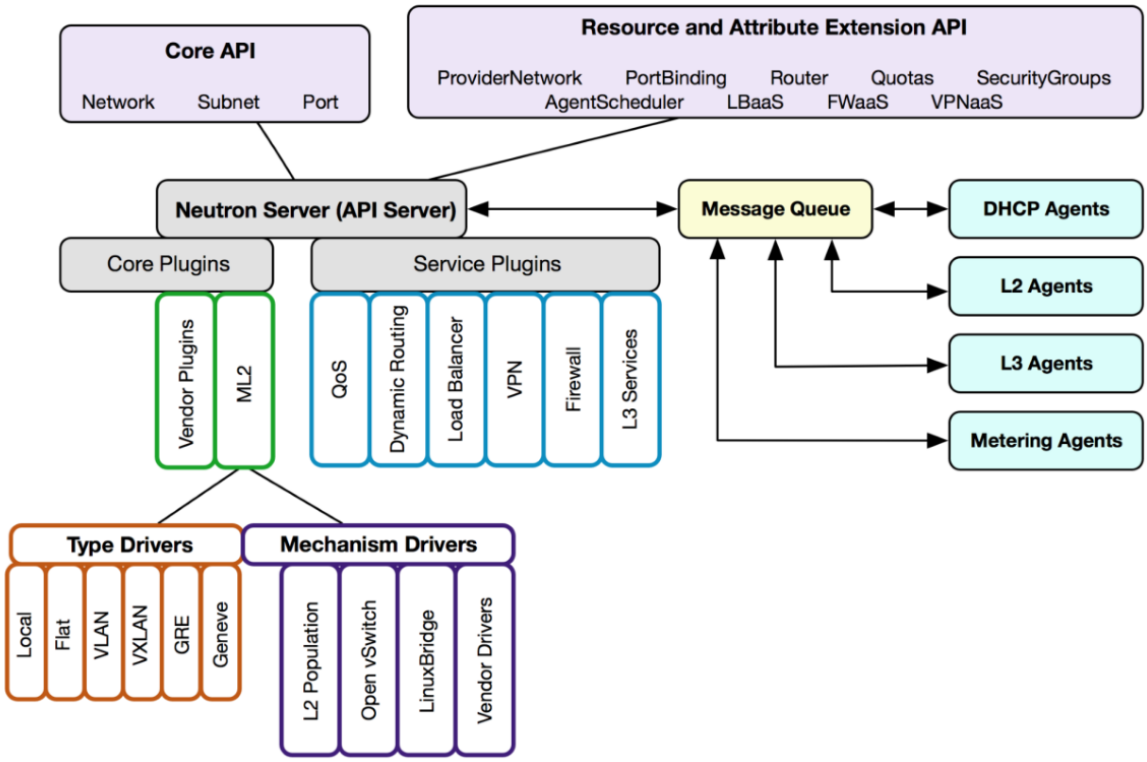
u pravilu koristi za manja testiranja. *Flat* mreža je mreža bez 802.1q VLAN tag oznake. To je *access* mreža. Ostali tipovi mreža su objašnjeni ranije u radu.

Drugi tip drivera je *„Mechanism driver“* koji je zadužen za informacije koje mu ispostavlja *Typedriver* i na temelju tih informacija osigurava da su mreže ispravno konfigurirane. Postoje tri tipa ovakvih drivera:

1. *Agent-based* – Linux bridge, Open vSwitch…
2. *Controller based* – Cisco ACI, VMware NSX…
3. *Top*-of-Rack – Cisco Nexus, Arista…

Neutron koristi agent bazirane Linux Bridge i Open vSwitch drivere i oni će biti obrađeni u ovome radu. Linux Bridge podržava lokalne, *flat,* VLAN i VXLAN mreže, Open vSwitch dodatno NVGRE i GENEVE. Ovi *driveri* omogućuju i limitiranje *broadcast* L3 prometa koji se prenosi preko *overlay* meže kada se koristi VXLAN. U normalnim okolnostima, *unicast, multicast* i *brodcast* promet (skraćeno, BUM promet) će biti nekontrolirano proslijeđen prema svim VTEP čvorovima. To može imati negativne posljedice za propusnost mreže, pa je poželjno kontrolirati takav promet mehanizmima koji omogućuju da se šalje promet koji je zaista potreban. Primjerice, *Broadcast* ARP zahtjev neće biti proslijeđen na VTEP čvor na koji nema *hostova* koji odgovaraju ARP zahtjevu. U Neutron se takva funkcionalnost naziva ARP *proxy*.

Sljedeći dijagram, Slika 21, pokazuje arhitekturu ML2 plugina, odnosno kako Neutron API komunicira s raznim priključcima i agentima zaduženima za izgradnju virtualne i fizičke mreže:



Slika 21 Neutron API i ML2[44]

Na dijagramu je prikazan interakcija između Neutron API-a, Neutron *plugina*, *drivera* i servisa (L2 i L3 agenti).

3.2.4. Network Namespace

Network namespace je način implementacije odvajanja mrežnih prostora između *tenanta*. To je logička kopija mrežnog *stacka* s vlastitom tablicom usmjeravanja, vatrozidima i mrežnim sučeljima. Svaka implementirani mrežni servis nalazi se u određenom *namespaceu*. Zahvaljujući tome, Neutron može npr. izolirati DHCP i *routing* servise za svaku mreži, omogućujući korisnicima kreiranje preklapajućih mreža s drugim korisnicima i drugim ili čak i istim *tenantima*. To omogućuje OpenStacku pružanje cjelokupnog doživljaja vlastitih računalnih i mrežnih resursa svakom korisniku.

Pogledajmo tri tipa *namespacea*:

- DHCP namespace - *qdhcp*

[44] James Denton: Learning OpenStack Networking, 2018., Fig 3.1

- „*Qdhcp*“ namespace sarži DHCP servis koji pruža IP adrese virtualnim mašinama koristeći DHCP protokol. „*dnsmasq*“ je proces koji obrađuje DHCP zahtjeve. „*qdhcp*“ ima sučelje priključeno u virtualni *switch* i može komunicirati s instanca VM-a u istoj mreži. *„qdhcp“* je kreiran za svaku mrežu gdje je uključen DHCP.

- Router namespace – *qrouter*
  - Predstavlja virtualni usmjernik i zadužen je za usmjeravanje mrežnog prometa između VM-ova u *subnetima* na koji je spojen. Kao i *qdhcp,* spojen je na jedan ili više virtualnih *switcheva*. U nekim slučajevima, više *namespacea* se može koristiti za izradu mrežne infrastrukture. To su „fip“ i „snat“ tipovi *namespacea* koji se koristi prilikom implementacije virtualnih ditribuiranih *routera* (DVR), izvan opsega rada.
- *Load Balancer namespace – qlbaas*
  - Predstavlja virtualni *load balancer* i servise poput HAProxyija. Spojen je isto kao i qdhcp i qrouter na virtualni *switch* .

Ovi tipovi *namespacea* se pojavljuju na čvorovima koji pokreću Neutron DHCP, L3 i LBaaS agente. Takvi servisi se tipično konfiguriraju na čvoru kontrolera i dediciranim mrežnim čvorovima. U slučaju DVR-a, možemo ih imati i *compute* čvorovima.

Neutron pruža nadogradivu *plugin* arhitekturu koja omogućuje implementaciju velikog broja mrežnih mogućnosti. Neutron održava logičku mrežnu arhitekturu i bazu podataka, a *pluginovi* i agenti na svakom čvoru su zaduženi za konfiguraciju virtualnih uređaja. ML2 *plugin* omogućuje razvoj novih korisnih funkcionalnosti.

4. Implementacija OpenStack okruženja

Laboratorij koji ću koristiti sastoji se od HP ProLiant DL160 G6 servera s četverojezgrenim Intel Xeon procesorom i 28GB radne memorije. Na njemu je instalacija Linux Ubuntu 16.04.1 operacijskog sustava. Umjesto fizičkih servera za OpenStack komponente odlučio sam se za virtualne mašine. Kao virtualizacijsko rješenje koristit ću Linux KVM, pouzdano virtualizacijsko rješenje za x86 hardverske sustave[45]. Uz to, KVM je *open source* i besplatan, pa je savršeno rješenje za naš *open source* cloud. Sustav će se sastojati od 5 virtualnih mašina, sljedećih naziva, Slika 22:

```
root@OPENSTACK1:/home/antun# virsh list
 Id    Name                           State
----------------------------------------------------
 1     controller01                   running
 2     compute01                      running
 3     compute02                      running
 4     compute03                      running
 5     network01                      running
```

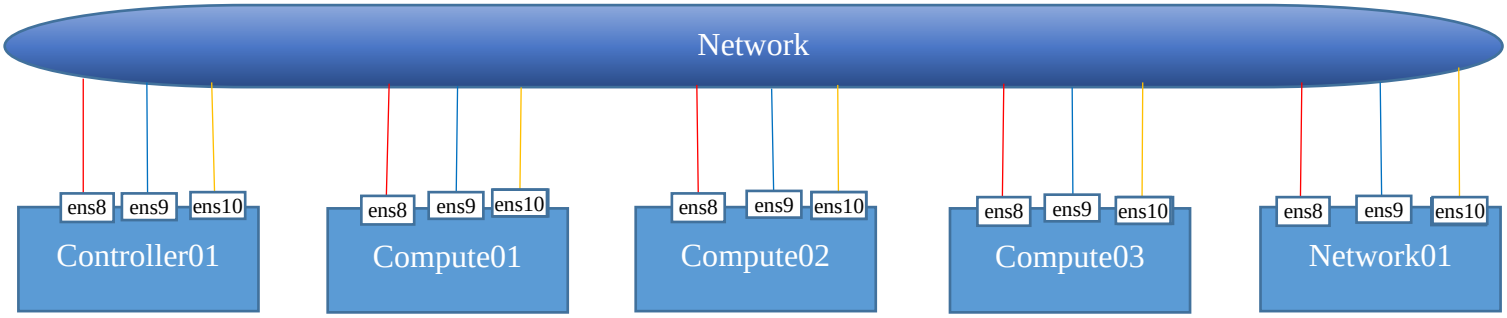
Slika 22 Nazivi virtualnih mašina

Sljedeće su funkcije:

1. Controller01 *node* pokreće Neutron server i mrežne servise, Horizon kao kontrolnu ploču, Keystone kao sustav identiteta, Glance za pohranu *imagea* i bazu podataka za sve servise
2. Compute01 pokreće Nova Compute servis i Linux Bridge mrežnog agenta
3. Compute02 pokreće Nova Compute servis i Open vSwitch mrežni agent
4. Compute03 pokreće Nova Compute servis i Open vSwitch mrežni agent
5. Network01 pokreće Open vSwitch agenta i L3 agenta

Svaka virtualna mašina ima tri mrežna adaptera. Prvi mrežni adapter je *management* adapter i za mrežnu komunikaciju između servisa (API). Drugi mrežni adapter služi za VXLAN *overlay* komunikaciju između čvorova. Treći mrežni adapter služi za VLAN komunikaciju između čvorova, Slika 23.

[45] https://www.linux-kvm.org/page/Main_Page



Slika 23 Mrežna shema laboratorija

Popis mreža, Tablica 2:

Mreža	Adresa	Adapter
Management	192.168.88.0/24	ens8
Overlay	10.20.0.0/24	ens9
VXLAN	10.60.0.0/24	*Tenant* network

Tablica 2 Popis mreža

Sljedeća tablica pokazuje IP adrese sučelja na virtualnim mašinama, Tablica 3.

Hostname	Sučelje	IP adresa
Controller01	ens8	192.168.88.11
	ens9	10.20.0.11
	ens10	L2 trunk
Compute01	ens8	192.168.88.21
	ens9	10.20.0.21
	ens10	L2 trunk
Compute02	ens8	192.168.88.22
	ens9	10.20.0.22

	ens10	L2 trunk
Compute03	ens8	192.168.88.23
	ens9	10.20.0.23
	ens10	L2 trunk
Network01	ens8	192.168.88.31
	ens9	10.20.0.31
	ens10	L2 trunk

Tablica 3 Popis mrežnih sučelja

Sučelja ens8 dobivaju IP adresu DHCP-om od *routera*, sučelje ens9 ima statičku IP adresu, a sučelje ens10 je podešeno bez IP adrese, kao L2 *trunk* sučelje, Slika 24:

```
root@controller01:/home/antun# cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
auto ens8
iface ens8 inet dhcp
auto ens9
iface ens9 inet static
        address 10.20.0.11
        netmask 255.255.255.0
auto ens10
iface ens10 inet manual
```

Slika 24 Konfiguracija mrežnog sučelja na Controlleru01

4.1. Osnovna konfiguracija

Instalaciju započinjemo na Compute01 čvoru, instaliranjem OpenStack *command-line* klijenta:

```
apt install python-OpenStackclient
```

Nakon toga instaliramo MySQL bazu i *messaging* servis:

```
apt install mariadb-server python-pymysql
```

```
apt install rabbitmq-server
rabbitmqctl add_user OpenStack rabbit
rabbitmqctl set_permissions OpenStack ".*" ".*" ".*"
```

Memcached se koristi za keširanje često korištenih podataka u RAM kako bi se poboljšale performanse sustava:

```
apt install memcached python-memcache
```

Krećemo s instalaciju prvog OpenStack servisa, a to je Keystone. Za njega moramo prvo napraviti bazu i podesiti prava:

```
mysql
CREATE DATABASE keystone;
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' IDENTIFIED BY 'keystone';
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY 'keystone';
quit;
```

Sada možemo instalirati Keystone servis i napraviti podešenja za bazu i token u konfiguraciji:

```
    # apt install keystone apache2 libapache2-mod-wsgi
```

```
/etc/keystone/keystone.conf
[database]
...
connection = mysql+pymysql://keystone:keystone@controller01/keystone

[token]
...
provider = fernet
```

Radimo popunjavanje Keystone baze i inicijalizaciju Fernet token repozitorija:

```
/bin/sh -c "keystone-manage db_sync" keystone
keystone-manage fernet_setup
```

```
 --keystone-user keystone --keystone-group keystone
   keystone-manage credential_setup
 --keystone-user keystone --keystone-group keystone
```

Nakon toga bootstrap servisnog kataloga:

```
keystone-manage bootstrap --bootstrap-password OpenStack
      --bootstrap-admin-url http://controller01:35357/v3/
      --bootstrap-internal-url http://controller01:5000/v3/
      --bootstrap-public-url http://controller01:5000/v3/
      --bootstrap-region-id RegionOne
```

Konfiguriramo Apache na controlleru01:

```
    sed -i '1s/^/ServerName controller01\n&/' /etc/apache2/apache2.conf
    systemctl restart apache2
```

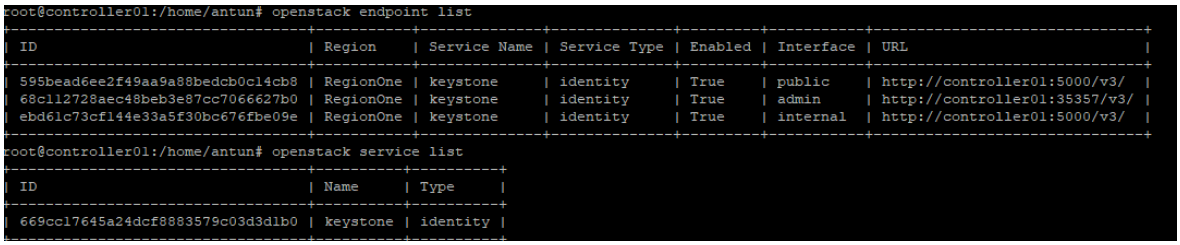
i podešavamo *environmental* varijable kako ne bismo morali upisivati kredencijale kod svake OpenStack naredbe:

```
# cat >> ~/adminrc <<EOF
     export OS_PROJECT_DOMAIN_NAME=default
     export OS_USER_DOMAIN_NAME=default
     export OS_PROJECT_NAME=admin
     export OS_USERNAME=admin
     export OS_PASSWORD=OpenStack
     export OS_AUTH_URL=http://controller01:35357/v3
     export OS_IDENTITY_API_VERSION=3
EOF
```

Nakon toga možemo pozvati sljedeću naredbu kako bismo učitali datoteku adminrc u *environmental varijable:*

```
source ~/adminrc
```

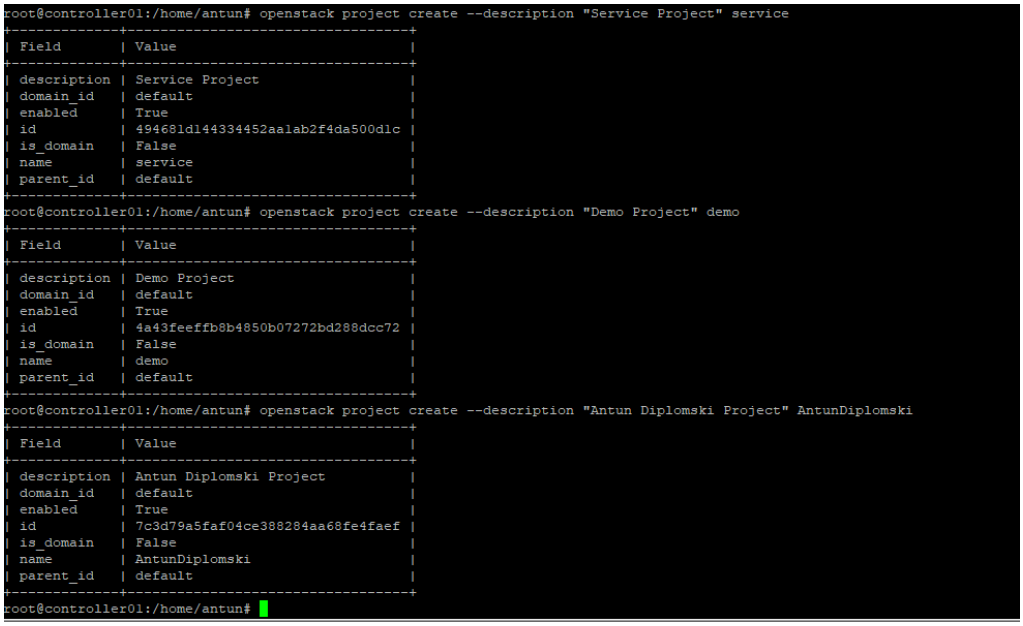
Svaki OpenStack servis koje se instalira mora biti registriran s Keystone servisom kako bi se servis mogao kontaktirati i nadzirati. Možemo vidjeti da smo za sada instalirali Keystone, Slika 25:



Slika 25 OpenStack Keystone potvrda instalacije

Sada možemo definirati korisnike, projekte/*tenante* i role u Keystoneu, provjera na Slika 25Slika 26.

```
# OpenStack project create --description "Service Project" service
# OpenStack project create --description "Demo Project" demo
# OpenStack project create --description "Antun Diplomski Project"
AntunDiplomski
```



Slika 26 Ispis naredbe za kreiranje projekta

```
OpenStack user create demo --password=demo
OpenStack user create antun --password=antun
OpenStack role add --project demo --user demo user
OpenStack role add --project AntunDiplomski --user antun user
```

Možemo nastaviti s instalacijom Glance servisa za pohranu *imagea* OS-a. Prilikom instalacije svake od OpenStack rola prvo se mora kreirati baza za dotičnu rolu:

```
mysql
CREATE DATABASE glance;
GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost' IDENTIFIED BY
'glance';
```

```
GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED BY 'glance';
quit;
```

U Keystoneu se zatim kreira korisnik za novu rolu:

```
OpenStack user create glance --domain default --password=glance
OpenStack role add --project service --user glance admin
```

I novi OpenStack servis:

```
# OpenStack service create --name glance \
--description "OpenStack Image" image
```

Na kraju pripreme za instalaciju servisa mora se pripremiti *endpoint* za API komunikaciju:

```
# OpenStack endpoint create --region RegionOne \
image public http://controller01:9292
# OpenStack endpoint create --region RegionOne \
image internal http://controller01:9292
# OpenStack endpoint create --region RegionOne \
image admin http://controller01:9292
```

Slijedi instalacija i podešavanje Glance servisa:

```
    # apt install glance
```

```
/etc/glance/glance-api.conf
/etc/glance/glance-registry.conf
[database]
...
connection = mysql+pymysql://glance:glance@controller01/glance
...
[keystone_authtoken]
...
auth_uri = http://controller01:5000
auth_url = http://controller01:35357
memcached_servers = controller01:11211
auth_type = password
user_domain_name = default
project_domain_name = default
project_name = service
username = glance
password = glance
```

```
/etc/glance/glance-api.conf
[paste_deploy]
...
```

```
flavor = keystone

[glance_store]
...
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images
```

```
/etc/glance/glance-registry.conf
[paste_deploy]
...
flavor = keystone
```

Popunjava se Glance baza podataka:

```
/bin/sh -c "glance-manage db_sync" glance
```

I skidamo prvi OS *image* „Cirros“ u Glance:

```
# OpenStack image create "cirros-0.4.0"
--file /tmp/images/cirros-0.4.0-x86_64-disk.img
--disk-format qcow2
    --container-format bare
--public
```

Provjera nam pokazuje da je *image* uspješno smješten u Glance, Slika 27:

```
root@controller01:/home/antun# openstack image list
+--------------------------------------+--------------+--------+
| ID                                   | Name         | Status |
+--------------------------------------+--------------+--------+
| 86283ac4-1fc3-4e06-8abc-a40d5521b35c | cirros-0.4.0 | active |
+--------------------------------------+--------------+--------+
root@controller01:/home/antun#
```

Slika 27 Cirros image u Glanceu

Slijedi instalacija Nova servisa za pokretanje virtualnih mašina. Nova se pokreće na *Compute* čvorovima, ali standardnu pripremu baze, korisnika i servisa radimo na Controlleru01:

```
mysql
CREATE DATABASE nova;
CREATE DATABASE nova_api;
CREATE DATABASE nova_cell0;
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' IDENTIFIED BY 'nova';
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' IDENTIFIED BY 'nova';
GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'localhost' IDENTIFIED BY 'nova';
GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%' IDENTIFIED BY 'nova';
```

```
GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'localhost' IDENTIFIED BY
'nova';
GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'%' IDENTIFIED BY 'nova';
quit;
```

```
# OpenStack user create nova --domain default --password=nova
# OpenStack user create placement
      --domain default --password=placement
# OpenStack role add --project service --user nova admin
# OpenStack role add --project service --user placement admin
```

```
# OpenStack service create --name nova
--description "OpenStack Compute" compute
    # OpenStack service create --name placement
      --description "Placement API" placement
```

Instaliranje i konfiguracija ostalih *compute* servisa koje koristi kontroler:

```
# apt install nova-api nova-conductor nova-consoleauth
nova-novncproxy nova-scheduler nova-placement-api
```

```
/etc/nova/nova.conf
[database]
...
connection = mysql+pymysql://nova:nova@controller01/nova

[api_database]
...
connection = mysql+pymysql://nova:nova@controller01/nova_api

[DEFAULT]
...
transport_url = rabbit://OpenStack:rabbit@controller01
...
my_ip = 10.10.0.100

[vnc]
...
enabled = true
vncserver_listen = 10.10.0.100
vncserver_proxyclient_address = 10.10.0.100

[api]
...
```

```
auth_strategy= keystone


[keystone_authtoken]
...
auth_uri = http://controller01:5000
auth_url = http://controller01:35357
memcached_servers = controller01:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = nova

[glance]
...
api_servers = http://controller01:9292

[oslo_concurrency]
...
lock_path = /var/lib/nova/tmp

[placement]
...
os_region_name = RegionOne
auth_url = http://controller01:35357/v3
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = placement
```

Popunjavanje baze:

```
/bin/sh -c "nova-manage api_db sync" nova
/bin/sh -c "nova-manage cell_v2 map_cell0" nova
/bin/sh -c "nova-manage cell_v2 create_cell
--name=cell1 --verbose" nova
/bin/sh -c "nova-manage db sync" nova
```

Nakon toga možemo preći na *Compute* čvorove i na njima izvršiti instalaciju i konfiguraciju Nova servisa:

```
apt install nova-compute
```

```
/etc/nova/nova.conf
[api]
...
auth_strategy= keystone

[keystone_authtoken]
...
auth_uri = http://controller01:5000
auth_url = http://controller01:35357
memcached_servers = controller01:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = nova

[DEFAULT]
...
transport_url = rabbit://OpenStack:rabbit@controller01
my_ip = 192.168.88.2x

[vnc]
...
vncserver_proxyclient_address = 192.168.88.2x
enabled = True
vncserver_listen = 0.0.0.0
novncproxy_base_url = http://controller01:6080/vnc_auto.html

[glance]
...
api_servers = http://controller01:9292

[oslo_concurrency]
...
lock_path = /var/lib/nova/tmp

[placement]
...
os_region_name = RegionOne
auth_url = http://controller01:35357/v3
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = placement
```

Na Slici 29 možemo provjeriti uspješnu instalaciju:

```
root@controller01:/home/antun# openstack compute service list
+----+------------------+--------------+----------+---------+-------+----------------------------+
| ID | Binary           | Host         | Zone     | Status  | State | Updated At                 |
+----+------------------+--------------+----------+---------+-------+----------------------------+
|  1 | nova-consoleauth | controller01 | internal | enabled | up    | 2019-08-28T02:07:31.000000 |
|  2 | nova-scheduler   | controller01 | internal | enabled | up    | 2019-08-28T02:07:31.000000 |
|  3 | nova-conductor   | controller01 | internal | enabled | up    | 2019-08-28T02:07:32.000000 |
|  8 | nova-compute     | compute01    | nova     | enabled | up    | None                       |
|  9 | nova-compute     | compute02    | nova     | enabled | up    | None                       |
| 10 | nova-compute     | compute03    | nova     | enabled | up    | None                       |
+----+------------------+--------------+----------+---------+-------+----------------------------+
root@controller01:/home/antun#
```

Slika 28 Provjera servisa

Slijedi instalacija Horizon Dashboard komponente na Controller01 koja pruža web sučelje za komunikaciju s OpenStack servisima:

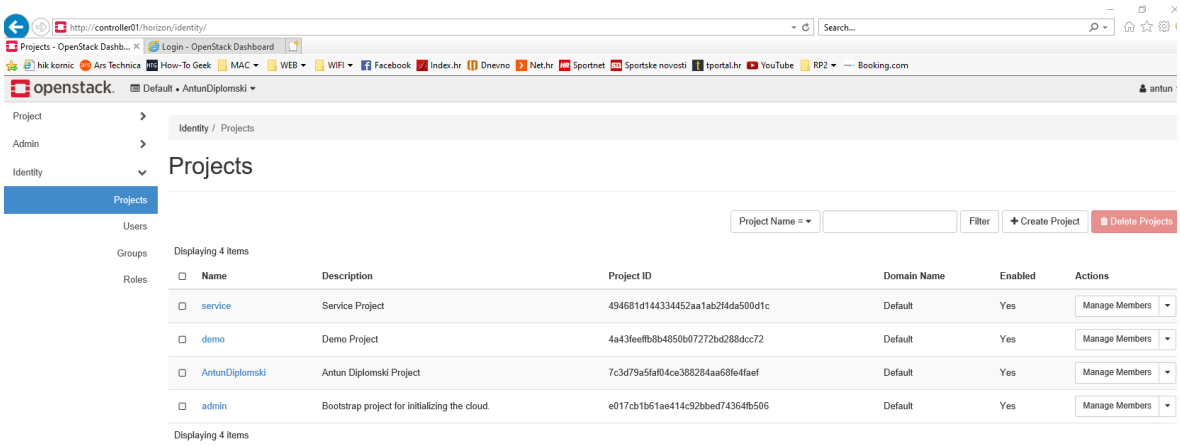
```
apt install OpenStack-dashboard
```

```
/etc/OpenStack-dashboard/local_settings.py
OPENSTACK_HOST = "controller01"
OPENSTACK_API_VERSIONS = {
    "identity": 3,
    "image": 2,
    "volume": 2,
}
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v3" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "Default"
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
OPENSTACK_NEUTRON_NETWORK = {
    'enable_router': False,
    'enable_quotas': False,
    'enable_ipv6': False,
    'enable_distributed_router': False,
    'enable_ha_router': False,
    'enable_lb': False,
    'enable_firewall': False,
    'enable_vpn': False,
'enable_fip_topology_check': False,
}
```

Podešavanje apachea za upsješno pokretanje Horizon web stranice:

```
gedit /etc/apache2/conf-available/OpenStack-dashboard.conf
WSGIApplicationGroup %{GLOBAL}
service apache2 reload
```

Nakon toga možemo napraviti provjeru i spojiti se Internet preglednikom na http://compute01/horizon: Slika 29.



Slika 29 *Horizon Dashboard*

Instalirali smo *OpenStack Identity, Image, Dashboard i Compute* servise. Preostala nam je još instalacija mreže, što ćemo napraviti u sljedećem poglavlju i tada je naš sustav sposoban za funkcioniranje, odnosno mrežnu komunikaciju između virtualnih mašina

4.2. Instalacija Neutrona i mrežna konfiguracija

U ovome poglavlju napravit ćemo instalaciju sljedećih komponenti:

- Neutron API server
- ML2 *plugin*
- DHCP agent
- Metadata agent

Neutron API server služi za komunikaciju agenata s kontrolerom, a Metadata agent za pružanje dodatnih informacija prilikom podizanja virtualnih mašina. ML2 *plugin* i DHCP agent su već objašnjeni u prethodnim poglavljima rada.

Priprema na kontroleru za instalaciju Neutrona:

```
mysql
CREATE DATABASE neutron;
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost' IDENTIFIED BY 'neutron';
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' IDENTIFIED BY 'neutron';
quit;
```

```
# OpenStack user create --domain Default --password=neutron neutron
# OpenStack role add --project service --user neutron admin
OpenStack service create --name neutron --description "OpenStack Networking" network

OpenStack endpoint create --region RegionOne
     network public http://controller01:9696
    # OpenStack endpoint create --region RegionOne
     network internal http://controller01:9696
    # OpenStack endpoint create --region RegionOne
     network admin http://controller01:9696
```

Instalacija Neutron paketa za kontroler:

```
# apt install neutron-server neutron-dhcp-agent
    neutron-metadata-agent neutron-plugin-ml2
python-neutronclient
```

Na svim ostalim čvorovima radimo instalaciju sljedećih paketa i konfiguriramo:

```
apt install neutron-plugin-ml2
```

`/etc/neutron/neutron.conf`

```
[database]
...
connection = mysql+pymysql://neutron:neutron@controller01/neutron
[DEFAULT]
...
```

```
auth_strategy = keystone
transport_url = rabbit://OpenStack:rabbit@controller01


[keystone_authtoken]
...
auth_uri = http://controller01:5000
auth_url = http://controller01:35357
memcached_servers = controller01:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = neutron
```

Potrebno je na svim čvorovima podesiti autentifikacijske parametre:

```
/etc/nova/nova.conf
[neutron]
...
url= http://controller01:9696
auth_url = http://controller01:35357
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = neutron
```

Neutron mora moći komunicirati s Nova servisom na kontroleru kako bi slao obavijesti o promjeni mrežne topologije:

```
/etc/neutron/neutron.conf
[nova]
...
auth_url = http://controller01:35357
auth_type = password
project_domain_name = default
```

```
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = nova
```

Na svim čvorovima podešavamo korištenje ML2 plugina:

`/etc/neutron/neutron.conf`

```
[DEFAULT]
...
core_plugin = ml2
```

Popunjavamo Neutron bazu na kontroleru:

```
/bin/sh -c "neutron-db-manage
--config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugins/ml2/ml2_conf.ini
upgrade head" neutron
```

Moramo još podesiti Metadata servis na kontroleru:

`/etc/nova/nova.conf`

```
[neutron]
...
service_metadata_proxy = true
metadata_proxy_shared_secret = MetadataSecret123
```

`/etc/neutron/metadata_agent.ini`

```
[DEFAULT]
...
nova_metadata_host = controller01
metadata_proxy_shared_secret = MetadataSecret123
```

Sada kada su svi OpenStack opći i OpenStack mrežni servisi instalirani i konfigurirani, preostaje nam još samo instalacija *Mechanism drivera* prije nego što možemo početi stvarati instance. Instalirat ćemo i Linux Bridge i Open vSwitch agente.

Podesit ćemo na DHCP-u MTU (*Maximum transmission unit*) na 1450 kako bismo bili sigurni da će *Overlay* VXLAN paketi proći na mreži pošto imaju *overhead* od 50 byteova.

```
/etc/neutron/dhcp_agent.ini
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
```

```
/etc/neutron/dnsmasq-neutron.conf
dhcp-option-force=26,1450
```

Možemo napraviti podešavanja ML2 plugina na kontroleru. Definiramo tipove mreža koje ćemo koristiti i parametre spajanja fizičkih s virtualnim adapterima:

```
/etc/neutron/plugins/ml2/ml2_conf.ini
[ml2]
...
type_drivers = local,flat,vlan,vxlan
mechanism_drivers = linuxbridge,l2population
tenant_network_types = vlan,vxlan

[ml2_type_vlan]
...
network_vlan_ranges = physnet1:40:43


[ml2_type_vxlan]
...
vni_ranges = 1:1000
```

Instaliramo Neutron Linux Bridge agent na controller01 i compute01:

```
# apt install neutron-plugin-linuxbridge-agent
```

Podesiti ćemo ens10 fizički adapter za VLAN mreže, i omogućiti VXLAN mreže na compute01 i controller01:

```
/etc/neutron/plugins/ml2/linuxbridge_agent .ini
[linux_bridge]
...
physical_interface_mappings = physnet1:ens10

[vxlan]
...
enable_vxlan = true
l2_population = true
arp_responder = true
local_ip = 10.20.0.X

[securitygroup]
...
firewall_driver = iptables
```

Povezivanje DHCP agenta i Linux bridgea na kontroleru:

```
/etc/neutron/dhcp_agent.ini
[DEFAULT]
...
interface_driver = linuxbridge
```

Nakon ovoga možemo napraviti provjeru i potvrditi da su svim mrežni agenti instalirani i pokrenuti, Slika 30.

```
root@controller01:/home/antun# openstack network agent list
+--------------------------------------+--------------------+--------------+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host         | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+--------------+-------------------+-------+-------+---------------------------+
| 212d7cb5-3e22-4154-a1fe-2eb7012d047b | Linux bridge agent | compute01    | None              | :-)   | UP    | neutron-linuxbridge-agent |
| 42fe6282-b54a-4f6d-be9c-1c795243d7d3 | DHCP agent         | controller01 | nova              | :-)   | UP    | neutron-dhcp-agent        |
| 522981a6-c472-4c1d-94e7-f9356fdaa271 | Metadata agent     | controller01 | None              | :-)   | UP    | neutron-metadata-agent    |
| e5dcc3a3-99c9-4de9-9d3a-4b500bd86c4e | Linux bridge agent | controller01 | None              | :-)   | UP    | neutron-linuxbridge-agent |
+--------------------------------------+--------------------+--------------+-------------------+-------+-------+---------------------------+
```

Slika 30 Provjera mrežnih agenata

Instalirali smo mreže komponente i Linux Bridge za čvorove Controller01 i Compute01. Sada možemo napraviti instalaciju i konfiguraciju Open vSwitch agenta na Compute02, Compute03 i Network01 čvorove. Prvo moramo na kontroleru podesiti ML2 da podržava OVS:

```
/etc/neutron/plugins/ml2/ml2_conf.ini
[ml2]
...
```

```
mechanism_drivers = linuxbridge,l2population,openvswitch
```

Zatim možemo instalirati OVS agent na Compute02, Compute03 i Network01:

```
apt install neutron-plugin-openvswitch-agent
```

i konfigurirati ga na sva tri čvora:

```
/etc/neutron/plugins/ml2/openvswitch_agent.ini
[agent]
...
tunnel_types = vxlan
l2_population = true
enable_distributed_routing = ...
vxlan_udp_port = 8472
arp_responder = true (nedostajalo)



[ovs]
...
integration_bridge = ...
tunnel_bridge = ...
local_ip = 10.20.0.x
bridge_mappings = physnet1:br-ens10


[securitygroup]
...
firewall_driver = iptables_hybrid
```

Konfiguriranje *bridgea* i dodavanje *porta* ens10 u *bridge* na compute2, compute3 I network01:

```
ovs-vsctl add-br br-ens10
ovs-vsctl add-port br-ens10 ens10
```

Potvrda uspješne konfiguracije *bridgea*, Slika 31.

```
root@compute02:/home/antun# ovs-vsctl list-br
br-ens10
br-int
```

Slika 31 Open vSwitch *bridge port*

Ako se spojimo na controller01 i pogledamo listu mrežnih agenata, možemo vidjeti da su se svi agenti podigli i prijavili kontroleru, Slika 32.

```
root@controller01:/home/antun# openstack network agent list
+--------------------------------------+--------------------+-------------+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host        | Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-------------+-------------------+-------+-------+---------------------------+
| 212d7cb5-3e22-4154-a1fe-2eb7012d047b | Linux bridge agent | compute01   | None              | :-)   | UP    | neutron-linuxbridge-agent |
| 3008a29d-d971-4f64-825e-83c3040ae6a4 | Open vSwitch agent | network01   | None              | :-)   | UP    | neutron-openvswitch-agent |
| 42fe6282-b54a-4f6d-be9c-1c795243d7d3 | DHCP agent         | controller01 | nova              | :-)   | UP    | neutron-dhcp-agent        |
| 522981a6-c472-4c1d-94e7-f9356fdaa271 | Metadata agent     | controller01 | None              | :-)   | UP    | neutron-metadata-agent    |
| 550243c5-c132-4b15-9815-6ca075c5b388 | Open vSwitch agent | compute03   | None              | :-)   | UP    | neutron-openvswitch-agent |
| 5ed973fb-7ee2-4360-ad8e-897fd70463f0 | Open vSwitch agent | compute02   | None              | :-)   | UP    | neutron-openvswitch-agent |
| e5dcc3a3-99c9-4de9-9d3a-4b500bd86c4e | Linux bridge agent | controller01 | None              | :-)   | UP    | neutron-linuxbridge-agent |
+--------------------------------------+--------------------+-------------+-------------------+-------+-------+---------------------------+
root@controller01:/home/antun#
```

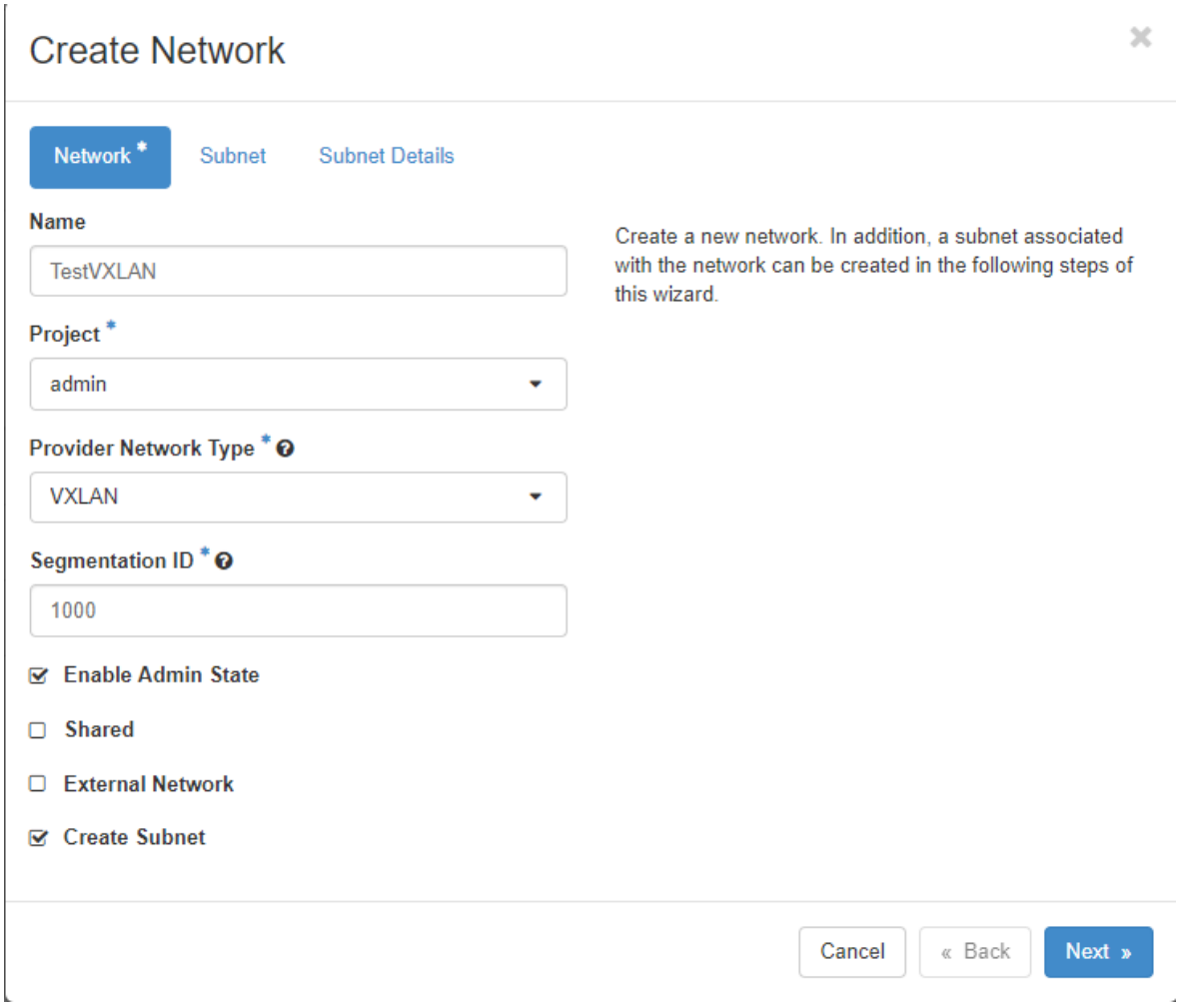
Slika 32 Lista mrežnih agenata

Instalirali smo OVS na 3 *nodea*, a dva imaju Linux Bridge. Oba drivera pružaju gotovo istu funkcionalnost spajanja virtualnih mašina na mrežu. OVS se oslanja na pravila tijeka (*flow rules*, engl.) za određivanje putanje prometa. Linux Bridge se oslanja na dobro poznate 802.1q standard, *bridge kernel module*, VLAN i VXLAN tagove za upravljanje prometom.

Idemo napraviti razne tipove mreža virtualne mašine i vidjeti kako se ponaša u produkciji.

4.3. Testiranje i mjerenja

Napraviti ćemo VXLAN mrežu sa *subnetom* 10.60.0.0/24, Slika 33:



Slika 33 Kreiranje VXLAN mreže

Na tu mrežu priključiti ćemo sljedeće virtualne mašine koje su napravljene na temelju ranije prikazanih „cirros“ *imagea*. VM-ovi imaju po jedan procesor, 1GB RAM-a i 1GB diska, Tablica 4:

Ime virtualne mašine	IP adresa	*Compute* čvor / ML2 driver
TestInstance1	10.60.0.9	Compute01 / Linux bridge
TestInstance2	10.60.0.5	Compute02 / OVS
TestInstance3	10.60.0.8	Compute01 / Linux bridge

Testinstance4	10.60.0.28	Compute03 / OVS
TestInstance5	10.60.0.24	Compute02 / OVS

Tablica 4 Popis VM-ova

Možemo vidjeti stvorene instance na Slici 35:

Project	Host	Name	Image Name	IP Address	Flavor	Status	Task	Power State	Time since created	Actions
admin	compute02	TestInstance5	cirros-0.4.0	10.60.0.24	tiny	Active	None	Running	50 minutes	Edit Instance
admin	compute03	TestInstance4	cirros-0.4.0	TestVLAN 10.40.0.101 TestVXLAN 10.60.0.28	tiny	Active	None	Running	1 week, 1 day	Edit Instance
admin	compute01	TestInstance3	cirros-0.4.0	TestVLAN 10.40.0.105 TestVXLAN 10.60.0.8	tiny	Active	None	Running	1 week, 1 day	Edit Instance
admin	compute02	TestInstance2	cirros-0.4.0	TestVLAN 10.40.0.125 TestVXLAN 10.60.0.5	tiny	Active	None	Running	1 week, 1 day	Edit Instance
admin	compute01	TestInstance1	cirros-0.4.0	TestVLAN 10.40.0.102 TestVXLAN 10.60.0.9	tiny	Active	None	Running	1 week, 1 day	Edit Instance

Slika 34 Cirrus instance

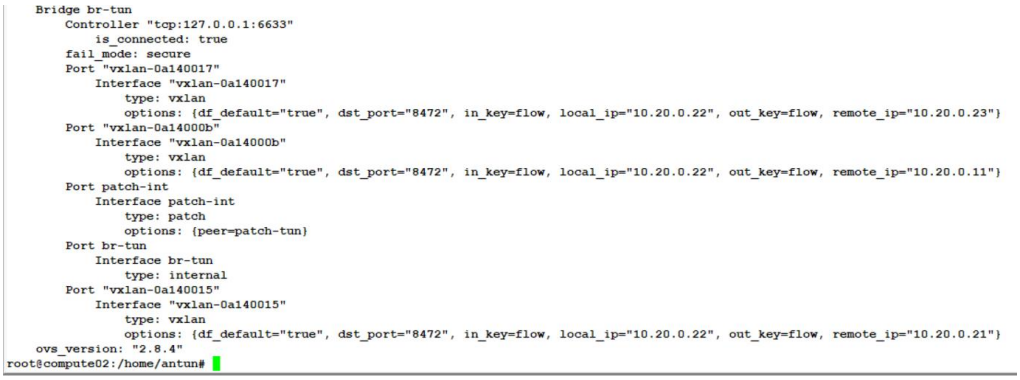
Spojimo se na prvu instancu i provjerimo možemo li mrežno komunicirati s drugima, Slika 35:

```
$ ping 10.60.0.5
PING 10.60.0.5 (10.60.0.5): 56 data bytes
64 bytes from 10.60.0.5: seq=0 ttl=64 time=3.375 ms
^C
--- 10.60.0.5 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 3.375/3.375/3.375 ms
```

```
$ ping 10.60.0.8
PING 10.60.0.8 (10.60.0.8): 56 data bytes
64 bytes from 10.60.0.8: seq=0 ttl=64 time=1.015 ms
64 bytes from 10.60.0.8: seq=1 ttl=64 time=1.298 ms
^C
--- 10.60.0.8 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.015/1.156/1.298 ms
```

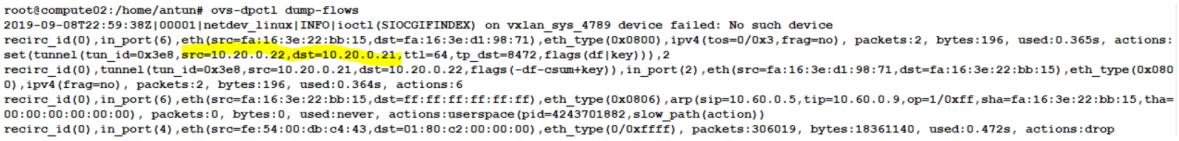
Slika 35 Provjera komunikacije

Komunikacija je uspješna što znači da VXLAN tuneli funkcioniraju. Upisivanjem naredbe „ovs-vsctl show“ na čvoru compute02 možemo vidjeti da se zatvaraju VTEP tuneli s čvorovima 10.20.0.11, 10.20.0.22 i 10.20.0.23, Slika 36:



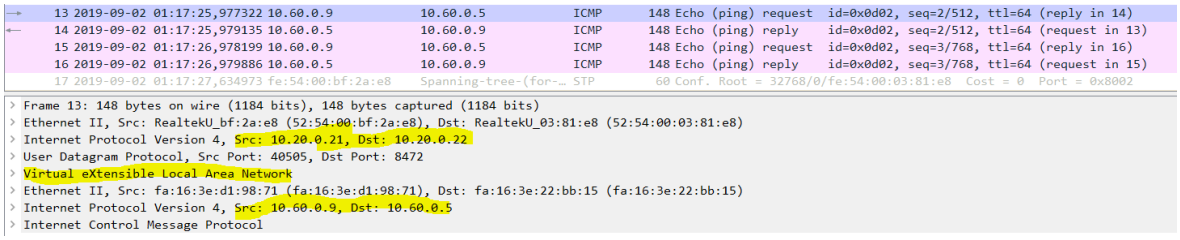
Slika 36 VXLAN VTEP tuneli

Ako pogledamo Openflow pravila koja se primjenjuju u trenutku komunikacije između 10.60.0.5 i 10.60.0.9 na controlleru02, možemo vidjeti kako se kreira pravilo za komunikacija između dva *nodea*, Slika 37:



Slika 37 OVS flow dump

VXLAN komunikaciju također možemo vidjeti i ako napravimo *capture* na mrežnoj kartici. Uzmimo primjer komunikacije između 10.60.0.9 i 10.60.0.5, Slika 38 .



Slika 38 Snimka VXLAN mrežnog prometa

Vidimo VXLAN IP paket koji putuje od 10.20.0.21 (compute01) do 10.20.0.22 (compute02). Compute02 prima paket, i skida VXLAN *header* unutar kojega je zapakiran ICMP paket od 10.60.0.9 do 10.60.0.5.

Ovime smo pokazali uspješnu VXLAN komunikaciju između dva čvora koja koriste različite SDN *switcheve* za implementaciju preklapanja (Linux Bridge i Open vSwitch). Također, pokazali smo kako se komunikaciju na drugom sloju odvija putem IP mreže, stvarajući IP

tunele po kojima se tunelira promet. Pogledajmo dalje performanse koje možemo ostvariti komunikacijom između čvorova i unutar čvora.

Napravit ćemo ping s 1000 byteova, Tablica 5.

Ping 1000 byteova, count 10, avg (ms)	10.60.0.8 / compute01 / Linux Bridge	10.60.0.28 / compute03 / OVS	10.60.0.24 / compute02 / OVS
10.60.0.5 / Compute02 / OVS	2,934	2,780	1,639
10.60.0.9 / compute01 / Linux Bridge	1,571	2,846	2,741

Tablica 5 Komunikacije 1000 *byteova*

Možemo zaključiti da u prijenosu velikih paketa nema razlike između OVS-a i Linux Bridgea ako se prenose između čvorova, dok je komunikacija unutar čvora ipak za milisekundu brža. Ovo su vrlo zadovoljavajuće brojke za L3 komunikaciju između čvorova u data centru.

Pogledajmo rezultate s manjim paketima, 10 *byteova*, Tablica 6:

Ping 10 byteova, count 10, avg (ms)	10.60.0.8 / compute01 / Linux Bridge	10.60.0.28 / compute03 / OVS	10.60.0.24 / compute02 / OVS
10.60.0.5 / Compute02 / OVS	2,772	2,632	1,544
10.60.0.9 / compute01 / Linux Bridge	1,314	2,683	2,724

Tablica 6 Komunikacije 10 byteova

Možemo dati jednaki zaključak kao i za prethodni slučaj, komunikacija na istom čvoru je nešto brža od komunikacije između čvorova.

Zadnji test će biti s paketima preko 1450 byteova (maksimalni MTU) da vidimo kako se naš labos ponaša kada mora raditi fragmentaciju IP paketa. Veličinu paketa ću postaviti na 20000 byteova, Tablica 7.

Ping 20000 byteova, count 10, avg (ms)	10.60.0.8 / compute01 / Linux Bridge	10.60.0.28 / compute03 / OVS	10.60.0.24 / compute02 / OVS
10.60.0.5 / Compute02 / OVS	9,106	8,905	3,495
10.60.0.9 / compute01 / Linux Bridge	2,629	9,250	8,421

Tablica 7 Komunikacije 20000 byteova

Ovdje možemo vidjeti dosta veće razlike u komunikaciji između čvorova gdje je razlika između latencije unutar čvora i do 6 milisekundi manja u odnosu na komunikaciju između čvorova. To možemo pripisati potrebi da se paketi moraju fragmentirati na postavljenih 1450 *byteova*. Ipak ovakve probleme ne očekujem u produkcijskim okruženjima gdje je MTU za VXLAN postavljen na 9000 *byteova* i nema potrebe za fragmentacijom paketa.

4.4. OpenStack vs VMware vRealize Automation

U ovome radu mogli smo vidjeti da nam cloud pruža zaista široke mogućnosti virtualne infrastrukture. Prilikom odluke o implementaciji cloud rješenja unutar tvrtke nekoliko faktora će biti ključno:

1. Stabilnost proizvoda
2. Mogućnosti
3. Podrška
4. Cijena

Cijenu sam namjerno stavio na zadnje mjesto jer iz vlastitog iskustva mogu reći da je cijena zadnje što profesionalna kompanija ima na kada želi implementirati novo rješenje.

Kao primjer konkurenta OpenStacku uzet ćemo vrlo popularni VMware vRealize Automation proizvod u tabličnom prikazu, Tablica 8.

	OpenStack	VMware vRealize Automation
Implementacija	Set zasebnih mikrokomponenti koje se instaliraju ovisno o potrebnim funkcionalnostima. Instalacija putem paketa na Linux OS, zasebna konfiguracija datoteka za svaki servis kroz CLI.	Set zasebnih komponenti za instalaciju i konfiguraciju. Mogućnost instalacije na Windows ili Linux OS. Relativno jednostavnija i vođena instalacija moguća kroz grafičko sučelje.
Kompleksnost	Modularna platforma s mogućnostima proširenja vlastitim kodom i podrška za više vrsta *hypervisora*, interoperabilnost s raznim NFV komponentama, ali s puno većom kompleksnosti konfiguracije i administracije.	Kompletan „*all-in-one*“ proizvod s ESXi kao jedinom opcijom hypervisora. Nemogućnost prilagođavanja komponenata vlastitim kodom. Manja kontrola platforme, ali veća jednostavnost administracije
Upravljanje	Horizon grafičko sučelje s mogućnošću proširenja 3d party alatima na Kontrolnom čvoru. Služi za dostup, podizanje i automatizaciju resursa. Mogućnost upravljanja kroz ugrađeni API. *Live* migracija moguća za KVN *hypervisore*.	Precizno grafičko sučelje s centralnim upravljanjem svim komponentama. vRealize Automation upravlja sustavom vCentera i ESXi-a. Puna podrška za *live* migracije mašina, i automatski monitoring. API se dodatno naplaćuje.

Zrelost	Open-source bez jasne budućnosti s velikim trenutnim zamahom i popularnošću među velikim tvrtkama. Neovisnost o jednom vendoru. Nepostojanje plaćene podrške i oslanjanje na zajednicu za rješavanje problema. Dokumentacija zna biti nepotpuna. Kraće postoji na tržištu od VMwarea.	Razvijen od nule počevši od *hypervisora*. Dobra dokumentiranost proizvoda. Podržava ga VMware kao jedan od svojih *core* proizvoda i izvora zarade i širenja na tržište. Koriste ga najveće svjetske firme radi vrlo dobre podrške i bez povijesti kritičnih problema.
Podrška i cijena	Podrška zajednice (commnity-support) uz mogućnost samostalnog razvoja. Individualne mogućnost podrške partnera. Certifikacija od strane partnera. Visoka cijena inicijalnog učenja sustava.	Podrška kroz sustav partnera ili direktno od VMwarea. Mogućnost certifikacije direktno od kompanije. Visoka cijena i kompleksan sustav licenciranja (single, per-processor, per-core, per-virtual-machine…).

Tablica 8 OpenStack vs VMware vRealize Automation osnovno[46]

[46] https://www.loomsystems.com/blog/single-post/2017/07/12/comparing-OpenStack-vs-vmware-vcloud

Zaključak

Softverski definirane mreže su postale stvarnost. Iz vlastitog iskustva mogu reći da i u Hrvatskoj sve više podatkovnih centara koristi SDN kao sustav umrežavanja resursa. Veliki vjetar u leđa dali su i *Cloud* sustavi poput OpenStacka jer je vrlo teško zamisliti ovako jednostavno upravljanje mrežama i mrežnim resursima u *Cloudu* bez SDN-a. VXLAN kao protokol za prijenos L2 podataka preko IP mreže za sada ima primat radi toga što ga, primjerice, Cisco, kao najpopularniji *vendor* mrežnih uređaja koristi u svojim rješenjima. Moja pretpostavka je da će tako i biti u sljedećih nekoliko godina do proširenja GENEVE protokola koji će pružiti interoperabilnost.

OpenStack kao *cloud* sustav pruža *open-source* rješenje za implementaciju *cloud* sustava za samostalno upravljanje korisnika virtualnom infrastrukturom. Sama instalacija je komplicirana, dugotrajna i opsežna radi potrebe za konfiguracijom i prilagođavanjem svake pojedine komponente sustava. Prednosti su mu to što je *open source*, pruža mogućnosti integracije s gotovim svim sustavima i podržava sva tri najpopularnija SDN protokola (VXLAN, NVGRE i GENEVE). Pruža mogućnosti integracije s KVM, VMware i Hyper-V *hypervisorima* što ga svrstava u najpoželjnije *private cloud providere* zbog mogućnosti integracije s postojećim „*in-house"* rješenjima. Također, veliki zajednica (engl. *community)* pruža podršku. No, upravo to bih naveo i kao najveću manu OpenStacka, nepostojanje pravog *supporta*. Red Hat , RackSpace i slične firme pružaju određene usluge vezane uz OpenStack, no u *enterprise* instalacijama najčešće se traži postojanje *support* tima od samog *vendora*. Kao drugi nedostatak naveo bih politiku promjene verzije svakih šest mjeseci. Svakih šest mjeseci OpenStack izbacuje novi „*release*", a postojeći prima podršku sveukupno 18 mjeseci. Enterprise kompanije često traže stabilno i dugoročno rješenje, poput Ubuntu LTS-a.

Iskustvo oko instalacije OpenStacka u laboratoriskom okruženju dovodi do zaključka da tvrtka koja kreće u implementaciju OpenStacka mora zaposliti tim inženjera školovanih za OpenStack koji će se baviti isključivo ovim cloud sustavom.

OpenStack je cjelovito cloud okruženje s velikim mogućnostima podrške raznih sustava, proširenja *pluginovima*, jednostavnim upravljanjem softverski definiranim mrežama, ali primjerice, u Hrvatskoj s ograničenim ljudskim resursima zaključio bih da će tvrtke radije

krenuti u implementaciju popularnijih sustava poput „VMware vRealize Automation“ *clouda*, najviše zbog stabilnosti, podrške i jasne vizije kompanije u budućnost sustava, bez obzira na početnu cijenu investicije.

Popis kratica

SDN Software Defined Network

VLAN Virtual Local Area Network

VXLAN Virtual eXtensible Local Area Network

NVGRE Network Virtualization using Generic Routing Encapsulation

STT Stateless Transport Tunneling

IP Internet Protocol

NAT Network Address Translation

TCP Transmission Control Protocol

NCP Network Control Protocol

LAN Local Area Network

IMP Interface Message Processor

STP Spanning-Tree Protocol

MTU Maximum Transmission Unit

DHCP Dynamic Host Configuration Protocol

OS Operating System

VM Virtual Machine

API Application Programming Interface

Popis slika

Slika 1 Staromodna mrežna arhitektura 1

Slika 2 SDN arhitektura 3

Slika 3 IMP 4

Slika 4 Token ring 5

Slika 5 Ethernet okvir 5

Slika 6 802.1q VLAN tag 6

Slika 7 VXLAN okvir 8

Slika 8 VXLAN tunel 9

Slika 9 NVGRE okvir 10

Slika 10 GENEVE okvir 12

Slika 11 Leaf spine moderna arhitektura preklopnika 13

Slika 12 Cisco ACI arhitektura 14

Slika 13 OpenStack[28] 19

Slika 14 Horizon Dashboard 21

Slika 15 Međuovisnost OpenStack komponenti 23

Slika 16 Neutron arhitektura 24

Slika 17 Dijagram Open vSwitch komponenti 26

Slika 18 Load balancer 27

Slika 19 VXLAN mesh 30

Slika 20 Neutron komponente 30

Slika 21 Neutron API i ML2 34

Slika 22 Nazivi virtualnih mašina 36

Slika 23 Mrežna shema laboratorija 37

Slika 24 Konfiguracija mrežnog sučelja na Controlleru01 38

Slika 25 OpenStack Keystone potvrda instalacije 41
Slika 26 Ispis naredbe za kreiranje projekta 41
Slika 27 Cirros image u Glanceu 43
Slika 28 Provjera servisa 47
Slika 29 *Horizon Dashboard* 48
Slika 30 Provjera mrežnih agenata 53
Slika 31 Open vSwitch *bridge port* 54
Slika 32 Lista mrežnih agenata 55
Slika 33 Kreiranje VXLAN mreže 56
Slika 34 Cirrus instance 57
Slika 35 Provjera komunikacije 57
Slika 36 VXLAN VTEP tuneli 58
Slika 37 OVS flow dump 58
Slika 38 Snimka VXLAN mrežnog prometa 58

Popis tablica

Tablica 1 Cisco ACI VMware NSX usporedba 16

Tablica 2 Popis mreža 37

Tablica 3 Popis mrežnih sučelja 38

Tablica 4 Popis VM-ova 57

Tablica 5 Komunikacije 1000 *byteova* 59

Tablica 6 Komunikacije 10 byteova 59

Tablica 7 Komunikacije 20000 byteova 60

Tablica 8 OpenStack vs VMware vRealize Automation osnovno 62

.

Literatura

[1] IETF, RFC 6071, Virtual eXtensible Local Area Network (VXLAN), https://tools.ietf.org/html/rfc7348, pristup srpanj 2019.

[2] IEEE, 802.1Q-2014 - IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks, 2014. https://ieeexplore.ieee.org/document/6991462, pristup srpanj 2019.

[3] RAJKO MIHAJLOVIĆ, *Upoznajte glavne prednosti SDN tehnologija*, 2017. https://www.ictbusiness.info/kolumne/upoznajte-glavne-prednosti-sdn-tehnologija, pristup srpanj 2019.

[4] NETWORK DIRECTION, *VXLAN Overview*, 2018. https://networkdirection.net/articles/routingandswitching/vxlanoverview/ pristup kolovoz 2019.

[5] MARGARET ROUSE, *NVGRE (Network Virtualization using Generic Routing Encapsulation)*, 2013. https://searchnetworking.techtarget.com/definition/NVGRE-Network-Virtualization-using-Generic-Routing-Encapsulation pristup kolovoz 2019.

[6] MARGARET ROUSE, *Generic Routing Encapsulation (GRE)*, 2011. https://searchnetworking.techtarget.com/definition/Generic-Routing-Encapsulation-GRE pristup kolovoz 2019.

[7] BRUCE DAVIE, *VXLAN, STT, and looking beyond the wire format*, 2013. https://octo.vmware.com/vxlan-stt-and-looking-beyond-the-wire-format/ pristup kolovoz 2019.

[8] BENJAMIN SHMAUS, *What is GENEVE?*, 2017. https://www.redhat.com/en/blog/what-geneve pristup kolovoz 2019.

[9] PALASH IJARI, *Comparison between Cisco ACI and VMWARE NSX*. IOSR Journal of Computer Engineering (IOSR-JCE), February 2017.

[10] BO HAN, VIJAY GOPALAKRISHNAN, LUSHENG JI, AND SEUNGJOON LEE, *Network Function Virtualization: Challenges and Opportunities for Innovations*. IEEE Communicatons Magazine, 2015.

[11] OPEN VSWITCH, *What Is Open vSwitch?*, http://docs.openvswitch.org/en/latest/intro/what-is-ovs/ pristup rujan 2019.

[12] NOVIFLOW, *The basics of SDN and the OpenFlow Network Architecture*, https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/ pristup rujan 2019.

[13] OPENSTACK OFFICIAL DOCUMENTATION, https://docs.OpenStack.org, pristup svibanj-rujan 2019.

[14] JAMES DENTON: *Learning OpenStack Networking*, Third Edition. PacktPub, 2018.

[15] LINUX KVM, https://www.linux-kvm.org/page/Main_Page, pristup rujan 2019.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu,

Antun Nemanić