UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sašo Kavčič

Orkestracija postavljanja virtualnih infrastruktur v oblaku

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2013

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01907/2013
Datum: 08.03.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **SAŠO KAVČIČ**

Naslov: **ORKESTRACIJA POSTAVLJANJA VIRTUALNIH INFRASTRUKTUR V OBLAKU**

ORCHESTRATING VIRTUAL INFRASTRUCTURE CREATION IN A CLOUD

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Na kratko opišite področje virtualizacije in računalništva v oblaku. Preučite platformo Openstack s poudarkom na storitvah, ki omogočajo orkestracijo postavljanja virtualnih infrastruktur in okolij, zlasti šablonske postavitve in avtomatsko skaliranje. Analizirajte problem postavljanja virtualne infrastrukture za izvajanje vaj pri računalniških komunikacijah in pripravite primere šablonskih postavitev s predpisano topologijo virtualne infrastrukture. Postavitve ustrezno pretestirajte in ovrednotite tudi druge možnosti uporabe orkestracijskih orodij.

Mentor:
M. Ciglarič
doc. dr. Mojca Ciglarič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Sašo Kavčič, z vpisno številko 24930397, sem avtor diplomskega dela z naslovom:

Orkestracija postavljanja virtualnih infrastruktur v oblaku

S svojim podpisom zagotavljam, da:

- sem diplomo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.), ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 14.06.2013 Podpis avtorja:

Zahvala

Za pomoč pri izdelavi diplomske naloge se zahvaljujem mentorici doc. dr. Mojci Ciglarič, asistentu dr. Matjažu Pančurju in osebju Laboratorija za računalniške komunikacije in omrežja. Zahvaljujem se tudi staršem za spodbudo na moji dolgi poti do diplome.

Kazalo

Seznam uporabljenih kratic

AMQP	(Advanced Message Queueing Protocol) odprt standarden komunikacijski protokol
API	(Application Programming Interface) aplikacijski programski vmesnik
AWS	(Amazon Web Services) množica Amazonovih storitev v oblaku
CPU	(Central Processing Unit) računalniška procesna enota, procesor
DHCP	(Dynamic Host Configuration Protocol) protokol za omrežno konfiguracijo računalnikov
DNAT	(Destination NAT) ponorno spreminjanje naslovov IP
DNS	(Domain Name System) sistem domenskih imen
EC2	(Amazon Elastic Compute Cloud) Amazonov sistem računalništva v oblaku
HTTP	(Hypertext Transfer Protocol) protokol za prenos informacij na svetovnem spletu
IaaS	(Infrastructure as a Service) storitveni model konfiguriranja infrastrukture v oblaku
IP	(Internet Protocol) internetni komunikacijski protokol
JSON	(JavaScript Object Notation) tekstovni odprti standard, temelječ na jeziku JavaScript
KVM	(Kernel-based Virtual Machine) virtualizacijska infrastruktura v jedru operacijskega sistema Linux
NAT	(Network Address Translation) proces spreminjanja naslovov IP
OS	(Operating System) operacijski sistem
PaaS	(Platform as a Service) storitveni model ponudbe programske platforme v oblaku
QCOW2	(QEMU Copy on Write verzije 2) format diskovne slike programa QEMU
QEMU	(Quick EMUlator) odprtokodni program za emulacijo in virtualizacijo računalnikov
REST	(Representation State Transfer) stil arhitekture programske opreme
RPC	(Remote Procedure Call) klic procedure med oddaljenima računalniškima sistemoma
S3	(Simple Storage Service) Amazonov servis shranjevanja podatkov v oblaku
SaaS	(Software as a Service) servisni model ponudbe programske opreme v oblaku
SNAT	(Source NAT) ponorno spreminjanje naslovov IP
SSH	(Secure Shell) protokol za varno upravljanje računalniških sistemov na daljavo
UUID	(Universally Unique Identifier) identifikacijski standard, uporabljen v programski opremi
VLAN	(Virtual Local Area Network) particija v lokalnem omrežju
VM	(Virtual Machine) navidezni računalnik
VMM	(Virtual Machine Monitor) nadzornik navideznih računalnikov

Povzetek

V diplomski nalogi je predstavljena rešitev hitrega kreiranja virtualne infrastrukture na platformi OpenStack s pomočjo OpenStack komponente Heat. Na hitro je predstavljena tematika storitev računalništva v oblaku in računalniške virtualizacije. Predstavljena je tudi platforma OpenStack. Podan je opis implementacije rešitve v obliki šablone Heat in skript v lupini Bash. Rešitev omogoča kreiranje virtualne infrastrukture uporabnika z enim ukazom, enostavno je tudi brisanje infrastrukture enega ali vseh uporabnikov. Za vsakega uporabnika se v procesu ustvarijo tri virtualne instance, ki so povezane v zasebno omrežje, prek usmerjevalnika pa tudi z zunanjim omrežjem. Prek sistema Heat se instance tudi konfigurirajo s potrebnimi programskimi paketi, kot sta bind9 in dnsutils. V procesu implementacije se je izkazalo, da Heat predstavlja močno orodje, ki lahko pripomore tudi k izvedbi učnega procesa. Ker je Heat za zdaj še v fazi razvoja, se je pokazalo tudi nekaj problemov, ki pa jih je bilo možno odpraviti. V delu je opisana tudi namestitev testnega okolja za preizkušanje razvite rešitve. Opisana je postavitev testnega okolja s pomočjo projekta Devstack. Na osnovi tega okolja je predstavljen tudi postopek preverjanja pravilnosti delovanja rešitve. V zaključku so predstavljene najpomembnejše ugotovitve pri implementaciji in možnosti za nadaljnji razvoj funkcionalnosti. Delu je priložena tudi izvorna koda v obliki šablone Heat in skript v lupini Bash. Komponenta Heat se je izkazala za zelo primerno orodje, ki pa bo še bolj uporabno, ko bo doseglo še večjo stopnjo zrelosti v naslednjih izdajah projekta OpenStack.

Ključne besede:

računalništvo v oblaku, OpenStack, OpenStack Heat, AWS CloudFormation, virtualizacija

Abstract

This diploma thesis presents a solution for quick deployment of virtual infrastructure on the OpenStack platform using OpenStack Heat. The topics of cloud computing and virtualization are briefly introduced at the start, as is the OpenStack platform. Given is also the description of the solution in the form of a Heat template and scripts implemented in Bash shell. The solution enables quick creation of tenant's virtual infrastructure with one command, deletion of tenant's infrastructure is made equally simple. In the creation process, for each tenant three instances are created, which are connected into a private network. A virtual router connects them to the supplied external network. Heat component is also used to configure instances with software packages, like bind9 and dnsutils. In the process of implementation, Heat turned out to be a powerful tool, which can also be used in the education process. Because Heat is still in the early stages of development, a few problems arose, but could be worked around. This work also describes the setup of testing environment used to test the solution. Testing environment is set up using the Devstack project. Based on this testing setup, the procedure for checking proper operation is also described. The conclusion presents the most important findings and ideas for further development. Enclosed is also the source code in the form of a Heat template and scripts in Bash shell. Heat proved to be a very suitable tool for the problem at hand, but usefulness is expected to be even greater, when the project reaches maturity in subsequent releases of OpenStack.

Keywords:

cloud computing, OpenStack, OpenStack Heat, AWS CloudFormation, virtualization

1 Uvod

1.1 Namen dela

Razvoj virtualizacijske tehnologije v zadnjih nekaj letih omogoča njeno uporabo tudi v učnem procesu. Namen dela je uporaba odprtokodne virtualizacijske platforme OpenStack in komponente Heat pri izvedbi vaje konfiguracije strežnika DNS. Izvedba izdelka pa mora biti dovolj splošna, tako da je z minimalnimi spremembami uporabna tudi za izvedbo drugih vaj. Prav tako mora biti dovolj enostavna za uporabo, ne sme pa zahtevati velike količine dodatnega ročnega dela, ki ni avtomatizirano. Omogočati mora vzporedno delo uporabnikov sistema, ob pogoju, da je zmogljivost računalniškega sistema zadostna.

V primeru, da se virtualizacija računalnikov pri izvedbi vaj ne bi uporabljala, obstajajo veliki problemi pri zagotavljanju strojne opreme. Vsak uporabnik zahteva vsaj en računalnik, ki ga mora v poteku dela konfigurirati, kot npr. strežnik DNS in po možnosti še en računalnik, ki se uporablja kot klient. Samo število računalnikov, ki so na voljo, je po navadi omejeno. Prav tako je treba računalnike pred izvedbo vaje pripraviti, na primer z namestitvijo operacijskega sistema in druge programske opreme. Po izvedbi vaje pa jih je prav tako treba pripraviti za izvedbo novih vaj, odstraniti trenutno konfiguracijo in podobno. Računalnike je treba povezati med sabo v računalniško omrežje, prav tako pa po možnosti poskrbeti za izolacijo le-teh iz varnostnih ali drugih razlogov.

Z uporabo platforme OpenStack je možna uporaba navideznih računalnikov, ki jih je zelo enostavno kreirati iz osnovne slike operacijskega sistema. Storitev Heat omogoča orkestracijo konfiguracije za posameznega uporabnika iz vnaprej pripravljene šablone. Pri tem se za posameznega uporabnika ustvarijo navidezni računalniki, povežejo se v navidezno lokalno omrežje, prav tako se jim omogoči dostop do zunanjega omrežja. Na navidezne računalnike se namesti v šabloni Heat navedena programska oprema (npr. strežnik DNS bind9). Vsi navedeni koraki se izvedejo z enim ukazom.

Z enim ukazom je prav tako možno brisanje ustvarjene virtualne infrastrukture enega uporabnika ali vseh uporabnikov.

1.2 Zgradba dela

V prvem poglavju so na splošno predstavljene računalniške storitve v oblaku, njihove osnovne karakteristike ter servisni in namestitveni modeli.

Drugo poglavje predstavlja tehnologijo virtualizacije ter različne virtualizacijske tehnike.

Sledi poglavje, kjer je predstavljena odprtokodna platforma OpenStack. Na kratko sta opisani zgodovina in konceptualna arhitektura projekta. Opisane so tudi posamezne komponente oziroma podprojekti, kot so Glance, Nova, Quantum in Heat.

Četrto poglavje predstavi implementacijo rešitve problema hitrega ustvarjanja virtualne infrastrukture. Predstavljene so zahteve implementacije, uporabljena tehnologija ter priprava slike operacijskega sistema. Prav tako sta predstavljena formata datotek JSON in format šablon Heat, sledi pa opis implementacije in zgradbe šablone. Opisan je tudi postopek priprave testnega okolja OpenStack s pomočjo projekta Devstack. Sledi opis skript v lupini Bash, ki se uporabljajo kot uporabniški vmesnik. Na koncu je opisan postopek preverjanja pravilnega delovanja rešitve.

V prilogi je prikaz šablone in implementiranih skript.

2 Računalniške storitve v oblaku

Računalništvo v oblaku v zadnjih letih doživlja precejšen razcvet, čeprav tehnologija in koncepti niso nič novega. A šele razmah širokopasovnih internetnih povezav in virtualizacije je omogočil tudi njegov prodor v množično uporabo pri podjetjih, raziskovalnih institucijah in seveda tudi pri končnih uporabnikih.

Računalništvo v oblaku je model, ki omogoča vsesplošen, priročen in samopostrežen omrežni dostop do množice računalniških virov (npr. omrežij, strežnikov, podatkovnih shramb, aplikacij in servisov), ki jih je možno zelo hitro pridobiti ali sprostiti, brez interakcije s ponudnikom storitev [1].

Ta model je sestavljen iz petih osnovnih karakteristik, treh storitvenih modelov in petih namestitvenih modelov.

2.1 Osnovne karakteristike[1]

- *Samopostrežba na zahtevo (angl. on-demand self-service)* Uporabnik lahko sam, brez interakcije z osebo, pridobi računalniške vire, kot so računska instanca, shramba podatkov itd.
- *Širokopasovni omrežni dostop (angl. broad network access)* Vse kapacitete so na voljo prek omrežnih povezav in prek standardnih mehanizmov dostopa na različnih platformah.
- *Množica virov (angl. resource pooling)* Na voljo je množica virov, ki se nato dinamično dodeljujejo v uporabo uporabnikom. Stranka po navadi ne ve in nima nadzora, od kod se ji določen vir alocira. Primeri virov so shramba, procesiranje in količina spomina.
- *Hitra elastičnost (angl. rapid elasticity)* Kapacitete je možno zelo hitro pridobiti ali jih sprostiti, v določenih primerih tudi samodejno, v odvisnosti od trenutnih potreb uporabnika ali stranke.
- *Merljivost storitve (angl. measured service)* Sistemi v oblaku optimizirajo delovanje z uporabo merjenja na določeni ravni abstrakcije. Uporaba virov se

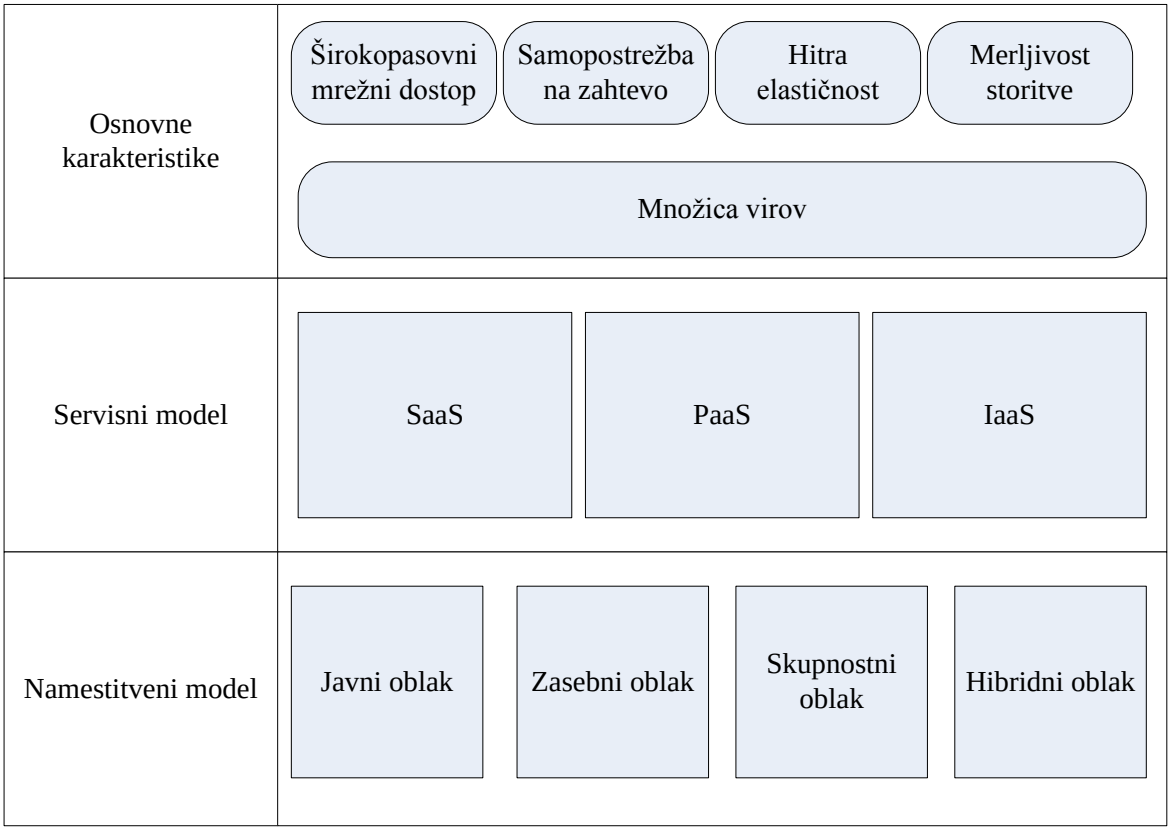
lahko meri, nadzira in sporoča, kar omogoča transparentnost tako za uporabnika kot ponudnika storitev.

2.2 Storitveni modeli oblakov

- *Softver kot storitev (angl. Software as a Service - SaaS)* – Uporabnik uporablja ponudnikove aplikacije, ki tečejo v oblaku. Aplikacija je na voljo prek spletnega ali programskega vmesnika. Uporabnik nima dostopa in ne upravlja infrastrukture. Primer je Googlova storitev gmail.
- *Platforma kot storitev (angl. Platform as a Service - PaaS)* – Uporabnik na ponudnikovo platformo lahko namesti svoje aplikacije. Te so kreirane s programskimi jeziki, knjižnicami ali orodji, ki jih ponuja ponudnik. Uporabnik ne upravlja ali nadzira spodaj ležeče infrastrukture.
- *Infrastruktura kot storitev (angl. Infrastructure as a Service - IaaS)* – V tem modelu uporabnik upravlja z osnovnimi viri, kot so navidezni računalniki, shramba podatkov, operacijski sistemi in aplikacije.

2.3 Namestitveni modeli oblakov

Modele oblakov lahko delimo tudi glede na način namestitve. Poznamo zasebne, skupnostne, javne in hibridne oblake. Pri zasebnem oblaku je infrastruktura oblaka na voljo uporabnikom določene organizacije in njenim poslovnim enotam. Skupnostni oblak je na voljo določeni skupnosti, upravlja pa ga določena organizacija iz te skupnosti. Javni oblaki so na voljo celotni javnosti, upravlja ga lahko poslovna, akademska ali druga institucija. Hibridni oblak je kombinacija dveh ali več prej naštetih vrst oblaka.



Slika 2.1 NIST-modeli računalništva v oblaku [2]

3 Virtualizacija

Virtualizacija je ena izmed glavnih tehnologij, ki omogočajo uspešno izvedbo storitev računalništva v oblaku. Omogoča kreiranje začasnih simuliranih računalniških virov (strojnih in programskih), kot so procesorske enote, računalniški spomin, shranjevalne enote in omrežni viri. Virtualizacija ustvari nivo abstrakcije med strojnimi viri in aplikacijami, ki jih uporabljajo [2].

Cilji virtualizacije so:

- *Čim večja izkoriščenost virov strojne opreme* S pomočjo razdelitve virov in deljenja procesorskega časa lahko dosežemo, da na določeni strojni opremi sobiva več storitev in bolj polno izkorišča opremo.
- *Centralizacija upravljanja virov na enem mestu* Večina virtualizacijskih platform ponuja vmesnike, kjer upravljamo, dodajamo in konfiguriramo vire na enem mestu.
- *Povečanje agilnosti, skalabilnosti in elastičnosti računalniških centrov* Virtualne naprave je mogoče konfigurirati v večini primerov precej hitreje kot pri delu direktno na strojni opremi. Določene platforme omogočajo tudi avtomatsko prekonfiguriranje glede na obremenjenost, zasedenost itd.
- *Pohitritev testiranja in diagnostike programske opreme* Virtualne računalniške sisteme je možno zelo hitro ustvariti, zagnati in brisati, kar omogoča hitrejšo izvedbo testov.
- *Izboljšanje prenosljivosti aplikacij in prenosljivost delovnega bremena* Omogočena je hitra selitev aplikacij med različnimi sistemi.
- *Zagotavljanje ločenosti med uporabniki za večji nivo zanesljivosti, varnosti in zasebnosti*

  Z uporabo virtualizacije so različni uporabniki popolnoma izolirani in ne morejo dostopati do virov drugih uporabnikov. To močno zmanjša možnost vplivanja uporabnikov na vire drugih, kar seveda poveča zanesljivost, varnost in zasebnost.
- *Strežniška konsolidacija* Z uporabo virtualizacije lahko na strojni opremi sobivajo storitve, ki brez nje ne bi mogle. To omogoča zmanjšanje količine strojne opreme in njeno večjo izkoriščenost.

- *Omogočanje samoupravljajoče platforme* Na osnovi meritev oziroma kriterijev je mogoče spreminjati število virtualnih računalnikov in prilagajati konfiguracijo glede na zahteve.

Virtualizacijsko okolje sestoji iz virtualnih računalnikov (*angl. virtual machine – VM),* virtualizacijske tehnologije za kreiranje virtualnih računalnikov in virtualnih naprav oziroma slik (*angl. virtual image, virtual appliance)*, ki tečejo na virtualnih računalnikih.

Zgodovina virtualizacije sega v leto 1965, ko so v podjetju IBM ustvarili prvi nadzornik virtualnih računalnikov (*angl. virtual machine monitor – VMM).* Prvi VMM je omogočal samo souporabo pomnilnika, naslednja verzija iz leta 1967 pa je že omogočala popolno virtualizacijo. Od takrat se virtualizacija hitro razvija. Dandanes obstaja več tehnologij virtualizacije, ki uporabljajo različne tehnike.

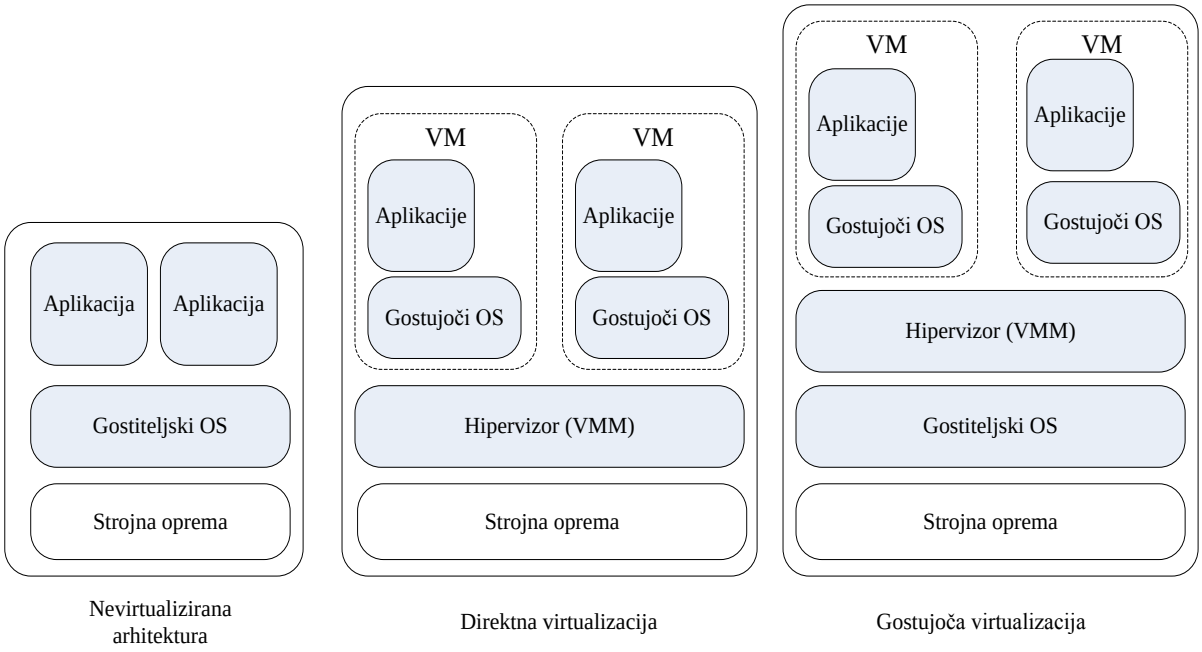
Kot že rečeno, je virtualizacija zelo pomemben del storitev računalništva v oblaku. Z uporabo virtualizacije je možno obvladati težave, ki se pojavljajo pri izgradnji in upravljanju računalniških centrov. V teh centrih si uporabniki delijo računalniške vire. Uporabniki imajo različne potrebe in zahteve po teh virih. Virtualizacija omogoča prilagajanje in alociranje teh virov, glede na uporabnikove zahteve. Najbolj znani virtualizacijski produkti so VMware, Oracle VirtualBox, Microsoft Hyper-V, Xen, KVM in Qemu. Večino podpira tudi platforma OpenStack.

3.1 Virtualizacijske tehnologije

Glede na lokacijo virtualizacijskega nivoja (*angl. hypervisor)* poznamo dve virtualizacijski arhitekturi: virtualizacija, kjer hipervizor teče direktno na strojni opremi (*angl. bare-metal virtualization),* in gostujoča virtualizacija *(angl. hosted).* Pri prvi je hipervizor nameščen direktno na strojni opremi, pri drugem tipu pa je hipervizor nameščen poleg operacijskega sistema.

Prednost direktne virtualizacije je v večji hitrosti. Mogoče je tudi razdeljevanje vhodno-izhodnih naprav v različne virtualne računalnike, kar omogoča direktni dostop do teh naprav. Hipervizor mora v tem načinu vsebovati gonilnike za vse naprave, prisotne na sistemu. Slaba stran je v težji namestitvi in konfiguraciji sistema.

Pri gostujoči virtualizaciji je hipervizor (VMM) nameščen na gostiteljskem operacijskem sistemu. Nameščanje hipervizorja je zato lažje. Slaba stran se lahko pokaže v slabših zmogljivostih, ker je treba vhodno-izhodne zahteve preusmeriti še skozi gostiteljski operacijski sistem.



Slika 3.1 Različne virtualizacijske arhitekture [2]

3.2 Virtualizacijske tehnike

Obstaja več tehnik za popolno virtualizacijo strojnih virov. Tehnike se razlikujejo po hitrosti virtualiziranih gostov in po tem, ali je potrebno spreminjanje gostujočega operacijskega sistema. Zaželena je tehnika virtualizacije, pri kateri spreminjanje gostujočega operacijskega sistema ni potrebno, a ima kljub temu dobre zmogljivosti izvajanja. Ti dve zahtevi se po navadi medsebojno izključujeta.

Poznamo tri tehnike virtualizacije:

- *Binarno prevajanje in domorodno izvajanje (angl. binary translation and native execution)* Pri tej tehniki se privilegirani ukazi v gostujočem operacijskem sistemu prevajajo, tako da ustrezajo delovanju v virtualiziranem sistemu. Direktno izvajanje takih ukazov ni dovoljeno, ker ima privilegije za nadzor strojne opreme

samo hipervizor. Ukazi uporabniškega nivoja se lahko izvajajo normalno, brez vmešavanja hipervizorja. Ta tehnika ima precej dobre zmogljivosti, prav tako pa se gostujočemu OS ni treba zavedati okolja, v katerem deluje. Je pa zahtevna pri gradnji podpore za binarno prevajanje privilegiranih ukazov. Primer so produkti iz družine VMware.

- *Paravirtualizacija (angl. paravirtualization)* Gostujoči OS je pri tej tehniki spremenjen, tako da se zaveda, da teče v virtualiziranem okolju. S klici komunicira s hipervizorjem in tako izvaja privilegirane strojne ukaze. Spremembe na gostujočem OS so enostavne za implementacijo, zato je ta način virtualizacije zelo učinkovit. Problem te tehnike pa je slaba kompatibilnost, saj ni mogoča virtualizacija OS, ki jih ni mogoče spremeniti (na primer Windows). Primer te tehnike je produkt Xen.
- *Strojno podprta virtualizacija (angl. hardware-assisted virtualization)* Proizvajalci procesorjev so z namenom pohitritve izvajanja virtualiziranih sistemov in povečanja kompatibilnosti začeli podpirati virtualizacijo na strojnem nivoju. Pri sistemih x86 sta na voljo tehnologiji Intel VT-x ter AMD-V. Pri tej tehniki se privilegirani ukazi gostujočih OS samodejno izvedejo v hipervizorju. Prevajanje teh ukazov je izvedeno v strojni opremi (CPU), zaradi česar je tudi hitrost izvajanja velika.

4 Platforma OpenStack

OpenStack je odprtokodni projekt, ki zagotavlja računalniške storitve v oblaku, po servisnem modelu IaaS (infrastructure as a service). Projekt upravlja fundacija OpenStack, ki je bila ustanovljena leta 2012. Projektu je danes pridruženih veliko podjetij, kot so Intel, AMD, Canonical, Red Hat, Cisco, Dell, HP in IBM. Projekt je sestavljen iz več podprojektov, ki skrbijo za nadzor nad računskimi storitvami, shrambo podatkov in omrežnimi viri, vse skupaj pa se nadzira prek nadzorne plošče, ki je dostopna prek spletnega vmesnika. Izdaje novih verzij programske opreme si sledijo vsakih šest mesecev. Trenutna verzija ima kodno ime Grizzly. Imena izdaj si sledijo po abecednem redu: Austin, Bexar, Cactus, Diablo, Essex, Folsom, Grizzly. Ime naslednje izdaje je Havana.

Zgodovina projekta OpenStack sega v leto 2010, ko sta NASA in Rackspace lansirala odprtokodno iniciativo računalništva v oblaku pod imenom OpenStack. Prva izvorna koda izvira iz Nasinega projekta Nebula in Cloud Files platforme podjetja Rackspace. Leta 2011 se je projektu pridružil tudi Canonical, podjetje, ki skrbi za razvoj Linuxove distribucije Ubuntu.

Misija projekta OpenStack je zagotavljanje skalabilnega in elastičnega računalništva v oblaku za javne in zasebne oblake, ki po velikosti segajo od najmanjših do največjih [3]. Implementacija oblakov mora biti enostavna, omogočati pa mora veliko skalabilnost.

Trenutno je v projektu OpenStack sedem komponent oziroma podprojektov v jedru. Dodatno obstaja še nekaj inkubacijskih podprojektov, ki so v primeru uspešne izvedbe vključeni v jedro projekta OpenStack.

Trenutni projekti v jedru:

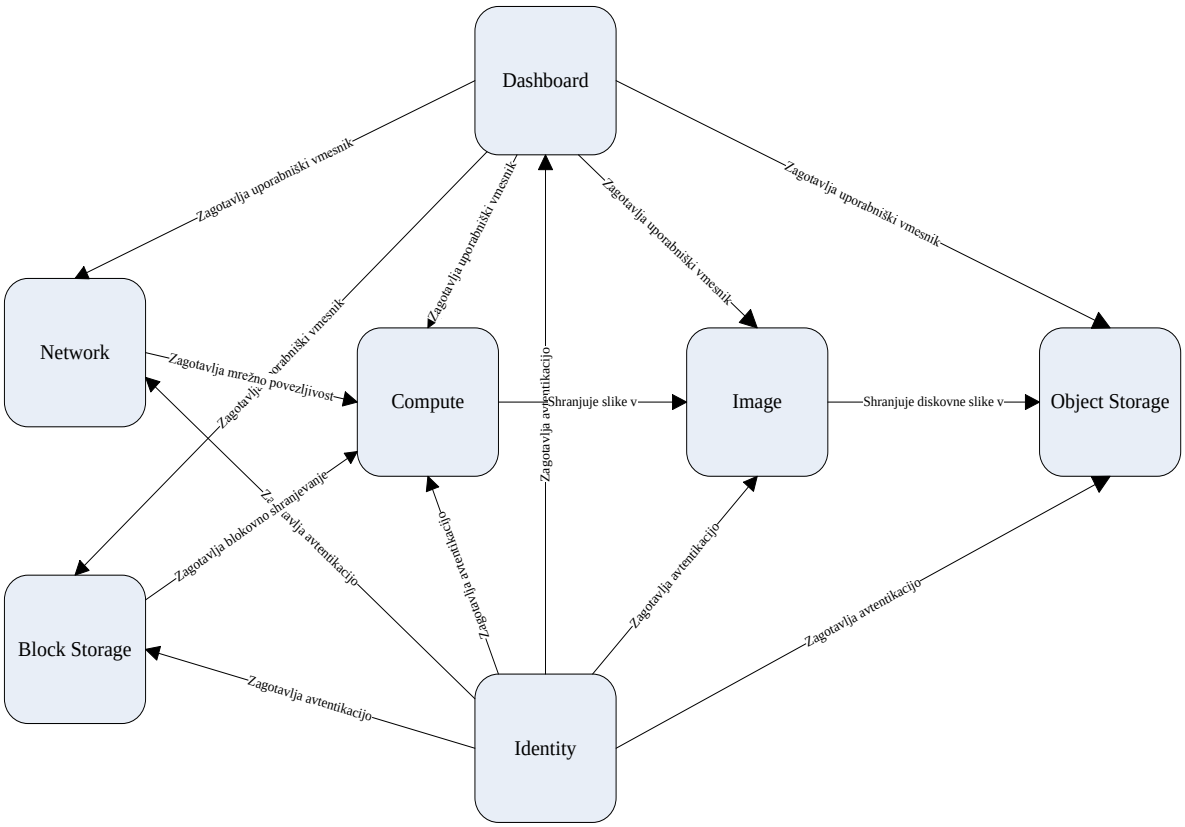
- *Block Storage (kodno ime Swift)* Shranjevanje datotek v obliki objektov.
- *Image (kodno ime Glance)* Shranjevanje slik operacijskih sistemov.
- *Compute (kodno ime Nova)* Razporejanje, zagon in brisanje virtualnih računalnikov in podporne aktivnosti.
- *Dashboard (kodno ime Horizon)* Nadzorna plošča implementirana prek spletnega vmesnika.
- *Identity (kodno ime Keystone)* Avtentikacija in avtorizacija uporabnikov.

- *Network (kodno ime Quantum)* Implementacija navideznih omrežij med virtualnimi računalniki in njihova povezava do zunanjosti.
- *Block Storage (kodno ime Cinder)* Implementacija virtualne shrambe v obliki blokovnih naprav, ki jih virtualni računalniki uporabljajo kot trajno diskovno shrambo.

Trenutni inkubacijski projekti:

- *Orchestration (kodno ime Heat)* Orkestracijski projekt, ki omogoča hitro kreiranje virtualne infrastrukture prek šablon.
- *Monitoring and Measurement (kodno ime Ceilometer)* Zagotavlja okvir za zbiranje, merjenje, opazovanje in sporočanje podatkov.

Dodatno je na voljo še mnogo drugih podpornih in neuradnih projektov. Spisek je dostopen na https://wiki.openstack.org/wiki/Projects. Dodatno je treba omeniti še projekt Devstack, ki omogoča hitro nameščanje okolja OpenStack, predvsem za testne ali razvojne namene.



Slika 4.1 Konceptualna arhitektura relacij med projekti OpenStack [3]

Kot je razvidno s slike 4.1, modul Identity zagotavlja avtentikacijo za vse ostale module, modul Dashboard pa zagotavlja spletni uporabniški vmesnik za krmiljenje vseh preostalih modulov. Modul Network zagotavlja omrežno povezljivost virtualnih računalnikov, ki jih ustvari modul Compute. Block Storage zagotavlja blokovno shranjevanje (navidezne diske) za virtualne računalnike modula Compute. Compute shranjuje slike za virtualne naprave v Image, ta pa jih lahko shranjuje v Object Storage. Pri namestitvi OpenStacka ni treba uporabiti vseh modulov. Modula Object Storage in Block Storage nista obvezna, če ne uporabljamo trajnih navideznih diskov in modul Image diskovnih slik ne shranjuje v Object Storage, ampak kar v datotečni sistem.

Upravljanje modulov je poleg spletnega vmesnika možno tudi prek vmesnika API v jeziku Python. Prav tako je možno prek ukazne vrstice z naslednjimi klienti: Nova, Glance, Cinder, Quantum, Keystone, Heat in Swift.

V nadaljevanju sledi podrobnejši opis modulov iz jedra projekta OpenStack: Nova, Glance, Cinder, Swift, Keystone, dodatno je bolj podrobno opisan tudi inkubacijski projekt Heat, ki predstavlja pomemben del rešitve.

4.1 Nova

Nova je najkompleksnejša in najbolj distribuirana komponenta platforme OpenStack in vsebuje veliko število procesov. Na osnovi zahtev API Nova kreira virtualne instance računalnikov. Glavni moduli so implementirani v jeziku Python.

Procesi vmesnika API:

- *nova-api* Sprejema zahteve API od končnih uporabnikov. Podpira vmesnike API OpenStack Compute, Amazonov EC2 in poseben administratorski vmesnik. Prav tako skrbi za kreiranje virtualnih instanc.
- *nova-api-metadata* Sprejema zahteve po meta podatkih od instanc. Na osnovi teh podatkov lahko instance izvedejo različne konfiguracijske korake.

Procesi jedra:

- *nova-compute* Proces skrbi za kreiranje in brisanje instanc na najnižjem nivoju. Uporablja vmesnike API različnih hipervizorjev, kot so XenApi za XenServer, libvirt za KVM ali QEMU ter VMwareAPI za VMware.
- *nova-schedule* Proces na osnovi zahteve za kreiranje instance in določenih kriterijev izbere računsko vozlišče, kjer bo instanca kreirana. Po izbiri poda zahtevo procesu nova-compute na tistem vozlišču.
- *nova-conductor* Modul, predstavljen v izdaji Grizzly. Deluje kot vmesnik med procesom nova-compute in podatkovno bazo.

Procesi omrežnega dostopa instanc:

- *nova-network* Proces, ki sprejema zahteve po omrežnem povezovanju instanc. Funkcionalnost se v zadnjih izdajah seli v omrežni modul Quantum, ki naj bi sčasoma popolnoma nadomestil nova-network.
- *nova-dhcpbridge* Skripta, ki skrbi za shranjevanje naslovov IP instanc v podatkovno bazo. Tudi ta funkcionalnost se seli v omrežni modul Quantum.

Procesi konzolnega dostopa:

- *nova-consoleauth* Demonski proces, ki skrbi za avtentikacijo uporabniških žetonov procesov nova-novncproxy in nova-xvpnvncproxy.
- *nova-novncproxy* Demonski proces, ki zagotavlja VNC dostop do instanc. Podpira novnc kliente, ki tečejo v spletnem brskalniku.
- *nova-xvpnvncproxy* Demonski proces, ki zagotavlja VNC dostop do instanc. Podpira java klienta, ustvarjenega posebej za OpenStack.
- *nova-cert* Demonski proces za upravljanje s certifikati po standardu x509.

Procesi za upravljanje z diskovnimi slikami:

- *nova-objectstore* Demonski proces, ki zagotavlja vmesnik S3 za registracijo diskovnih slik v slikovno shrambo Glance. Namenjen je predvsem namestitvam, ki uporabljajo orodja euca2ools.
- *euca2ools* Ta klient ni del okolja OpenStack, ga pa ta lahko podpira. Je zbirka ukazov v ukazni vrstici za upravljanje z viri v okviru storitev v oblaku. Je del projekta Eucalyptus in je odprtokodna implementacija ukazov Amazonovega okolja EC2.

Nova klient omogoča izvajanje ukazov v ukazni vrstici, lahko v imenu končnega uporabnika ali pa v imenu administratorja. Ukaz nova-manage je namenjen samo izvajanju administratorskih ukazov.

V delovanju se uporablja tudi vrsta *(angl. queue)*, ki predstavlja mehanizem za pošiljanje sporočil med različnimi procesi v sistemu. Vrste so implementacija standarda AMQP *(advanced message queueing protocol).* Najpogosteje uporabljena vrste je RabbitMQ, možno pa je uporabiti tudi Apache Qpid ali zeroMQ.

Vse nastavitve in stanja se v sistemu OpenStack shranjujejo v podatkovno bazo. Teoretično se lahko uporablja katerakoli podatkovna baza, ki jo podpira projekt SQLAlchemy, v praksi pa se največ uporabljata MySQL in PostgreSQL.

Nova uporablja mnoge druge servise OpenStack: Keystone za avtentikacijo, Glance za shranjevanje diskovnih slik in Horizon za spletni vmesnik.

4.2 Dashboard

Dashboard s kodnim imenom Horizon je modularna spletna aplikacija, implementirana na spletni platformi Django. Administratorjem in končnim uporabnikom omogoča dostop do servisov OpenStack. Arhitektura:

- Po navadi je nameščen na spletnem strežniku Apache prek modula `mod_wsgi`.
- Uporablja podatkovno bazo, ki je konfigurabilna, po navadi se uporablja MySQL. Za podatke po navadi uporablja druge servise, tako da sam shranjuje malo podatkov.

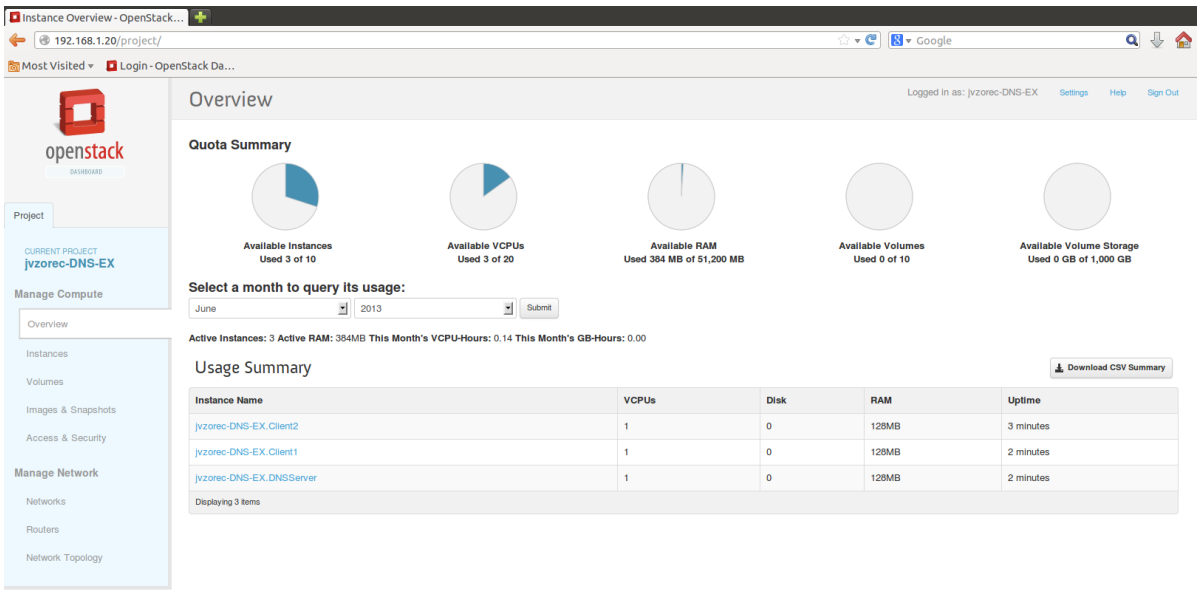
Spletni vmesnik je viden na sliki 4.2.

4.3 Glance

Glance je servis, ki omogoča shranjevanje slikovnih datotek. Sestavljajo ga štirje glavni deli:

- *glance-api* Sprejema zahteve API za shranjevanje, branje in odkrivanje slikovnih datotek.
- *glance-registry* Shranjuje, procesira in vrača meta podatke o slikah, kot sta velikost in tip slik.
- *Podatkovna baza za shranjevanje meta podatkov* Podobno kot pri Novi je možno izbrati podatkovno bazo za shranjevanje podatkov, po navadi se za to uporablja MySQL.
- *Shranjevalni repozitorij* Repozitorij za shranjevanje diskovnih slik. Glance podpira veliko vrst repozitorijev, kot so servis Swift, normalni datotečni sistem, naprave RADOS, Amazon S3 in HTTP. Repozitorij lahko omogoča tudi samo branje.

Glance sprejema zahteve API po branju slikovnih datotek od komponente Nova ali od končnih uporabnikov.



Slika 4.2 Pregledna stran spletnega vmesnika Dashboard

4.4 Keystone

Keystone je identifikacijski servis platforme OpenStack. Servis ima dve primarni funkciji:

- *Upravljanje z uporabniki* Skrbi za evidenco uporabnikov in evidenco uporabnikovih pravic.
- *Servisni katalog (angl. service catalog)* Zagotavlja katalog vseh razpoložljivih servisov in njihovih dostopnih točk na sistemu.

Termini v servisu Keystone:

- *Uporabnik* Digitalna reprezentacija osebe, servisa ali sistema. Uporabnik je dodeljen najemniku. Lahko mu je dodeljen tudi identifikacijski žeton.
- *Reference (angl. Credentials)* Podatki v lasti uporabnika (oziroma jih pozna le on), ki služijo za njegovo avtentikacijo. To so lahko uporabniško ime in geslo, avtentikacijski žeton itd.

- *Avtentikacija (angl. Authentication)* Avtentikacija je postopek preverjanja identitete uporabnika. Uporabnik najprej predstavi uporabniško ime in geslo, identifikacijski servis pa mu nato izda žeton, ki ga lahko uporabi v bodočih zahtevah.
- *Žeton (angl. token)* Avtentikacijska enota, ki uporabniku omogoča dostop do določenih virov. Ima omejen rok trajanja.
- *Najemnik (angl. tenant)* Skupek virov ali identifikacijskih objektov. Glede na servis je to lahko stranka, organizacija, račun ali projekt.
- *Servis* Servis platforme OpenStack, kot so Nova, Glance, Swift itd. Servis zagotavlja dostopno točko, do katere uporabniki dostopajo.
- *Dostopna točka (angl. endpoint)* Omrežno dostopen naslov, kjer je možen dostop do servisa.
- *Vloga (angl. role)* Osebnost, ki jo zavzame uporabnik pri opravljanju določene množice operacij. Vlogi pripadajo določene pravice in privilegiji. Uporabnik v določeni vlogi nasledi njene pravice in privilegije. Žeton, ki je izdan uporabniku, vključuje tudi seznam vlog, ki jih lahko zavzame. Interpretacija vlog je prepuščena posameznim servisom.

4.5 Cinder

Komponenta Cinder predstavlja blokovno shrambo platforme OpenStack. Blokovna shramba omogoča kreiranje trajnih virtualnih diskov, ki jih uporabniki lahko uporabljajo na virtualnih instancah. Instance uporabljajo začasno diskovno shrambo *(angl. ephemereal),* ki ob brisanju instance izgine. Blokovna shramba omogoča, da uporabnik ohrani podatke tudi po brisanju instance in jih na novi instanci uporablja dalje.

Procesi servisa Cinder:

- *cinder-api* Sprejema zahteve API in jih usmerja proti procesu cinder-volume.
- *cinder-volume* Vzdržuje podatkovno bazo virtualnih diskov, komunicira z drugimi procesi (npr. cinder-scheduler) in posreduje zahteve za blokovno shrambo strojni ali programski opremi. Podprtih je mnogo načinov za zagotavljanje shrambe, kot so IBM, SolidFire, NetApp, Nexenta, Zadara, GlusterFS, Linux iSCSI in drugi.

- *cinder-scheduler* Demonski proces za izbiro optimalne shrambe pri določeni zahtevi. Pri tem lahko upošteva kapacitete, območja razpoložljivosti, tip shrambe itd.
- *cinder-backup* Skrbi za shranjevanje varnostnih kopij v objektno shrambo (Swift).

Cinder omogoča poleg shranjevanja varnostnih kopij tudi izdelavo posnetkov shrambe *(angl. snapshot)*. V procesu shranjevanja varnostnih kopij se shranijo samo v shrambi obstoječi podatki, pri posnetku pa se naredi natančna slika celotne shrambe oziroma virtualnega diska. Možno je tudi določanje kvot pri uporabi shrambe. Komponenta Cinder je opcijska in njena uporaba ni nujna.

4.6 Swift

Swift je komponenta, ki zagotavlja objektno shrambo. Procesi servisa Swift:

- *swift-proxy-server* Sprejema zahteve prek vmesnika API ali prek vmesnika HTTP. Sprejema zahteve za nalaganje datotek, spremembe metapodatkov, generiranje kontejnerjev itd. Datoteke in vsebino kontejnerjev lahko posreduje tudi prek spletnih brskalnikov.
- *Računski strežniki (angl. account servers)* upravljajo z uporabniškimi računi v okviru objektne shrambe.
- *Kontejnerski strežniki (angl. container servers)* upravljajo z mapiranjem kontejnerjev (direktorijev) v okviru objektne shrambe.
- *Objektni strežniki (angl. object servers)* upravljajo z objekti (datotekami).
- Preostali procesi se ukvarjajo z replikacijo, revizijo, posodabljanjem in brisanjem starih podatkov.

Identifikacijo in avtentikacijo v okviru objektne shrambe po navadi opravlja identifikacijska komponenta platforme OpenStack Keystone. Dostop do objektne shrambe je mogoč tudi programsko prek jezikov PHP, Python, Java, C#/.NET in Ruby.

Objektna shramba Swift je opcijska komponenta, ki je v namestitvi platforme OpenStack ni nujno uporabiti. Komponenta prav tako ne igra vloge pri izvedbi tega dela.

4.7 Heat

Servis Heat je namenjen orkestraciji aplikacij v oblaku v okviru projekta OpenStack. Uporablja šablone storitve AWS CloudFormation podjetja Amazon. Orkestracija poteka prek REST vmesnika API platforme OpenStack ali prek vmesnika API, združljivega s storitvijo CloudFormation. Šablone omogočajo ustvarjanje večine virov OpenStack, kot so instance, plavajoči naslovi IP, varnostne grupe itd. Omogočene so tudi bolj napredne funkcije, kot je samodejno skaliranje aplikacij *(angl. autoscaling).* Cilj razvoja te komponente je predvsem v tem, da bo tudi na platformi OpenStack na voljo funkcionalnost, ki jo že ima platforma Amazon Web Services. Heat je integriran z identifikacijskim modulom Keystone in je, tako kot večina drugih komponent v platformi OpenStack, implementiran v jeziku Python. Heat ima štiri glavne dele:

- *heat* Orodje heat je ukaz v ukazni vrstici, ki komunicira prek vmesnika API s *heat-api* delom.
- *heat-api* Komponenta heat-api predstavlja OpenStack vmesnik REST API, ki sprejema zahteve in jih pošilja prek RPC komponenti heat-engine.
- *heat-api-cfn* Komponenta zagotavlja vmesnik API, združljiv z Amazonovim vmesnikom AWS CloudFormation. Prav tako zahteve pošilja prek protokola RPC komponenti heat-engine.
- *heat-egine* Komponenta heat-engine skrbi za kreiranje aplikacije oziroma sklada, tako da izvaja klice prek vmesnikov API drugih komponent.

Šablone so podrobneje opisane v petem poglavju o implementaciji rešitve.

4.8 Quantum

Mrežna komponenta OpenStacka s kodnim imenom Quantum je bila ustvarjena z namenom, da uporabnikom in administratorjem ponudi bogat vmesnik API, ki ga le-ti lahko uporabijo za kreiranje obsežne virtualne omrežne infrastrukture. V starejših izdajah OpenStack je bil v ta namen uporabljan modul nova-network. Pokazala pa se je potreba po bolj fleksibilni komponenti z večjim obsegom funkcionalnosti.

Quantum je servis virtualnih omrežij, ki ponuja obsežen vmesnik API. Ta definira omrežno povezljivost in naslavljanje naprav, ki jih ponuja na primer servis Nova. Vmesnik ponuja več abstrakcij, ki opisujejo omrežne vire:

- *Omrežje (angl. network)* Predstavlja izoliran omrežni segment L2. Ustreza VLAN-u v realnem svetu.
- *Podomrežje (angl. subnet)* Blok naslovov IP verzije 4 ali verzije 6 in pridruženega konfiguracijskega stanja.
- *Vrata (angl. port)* Priključna točka za priključitev naprave, kot je navidezni omrežni vmesnik navideznega računalnika. Opisuje tudi omrežno konfiguracijo, kot sta MAC in naslov IP.

Uporabnik konfigurira omrežja in podomrežja. Nato poveže virtualne računalnike z vrati na teh omrežjih. Vsak najemnik lahko ustvari večje število omrežij in izbira naslove IP neodvisno od drugih najemnikov, tudi če se naslovi IP prekrivajo. To omogoča napredno konfiguracijo omrežij in prenos realnih konfiguracij v virtualne.

Administrator lahko omogoči to napredno konfiguracijo uporabnikom ali pa ne. Tudi v drugem primeru Quantum zaradi svoje fleksibilnosti administratorju močno olajša delo.

Quantum za samo implementacijo omrežne povezljivosti uporablja vtičnike. Kot je opisano zgoraj, Quantum samo predpisuje abstraktne enote, ki se uporabljajo za povezovanje, ne predpisuje pa, kako je ta funkcionalnost implementirana. Sama implementacija je prepuščena vtičnikom. Trenutno Quantum podpira vtičnike Open vSwitch, Cisco, Linux Bridge, Nicira NVP, Ryu, NEC OpenFlow, Big Switch-Floodlight REST Proxy, PLUMgrid, Hyper-V, Brocade in Midonet. Vtičniki se razlikujejo glede podprte strojne opreme, značilnosti, podprtosti hipervizorjev, skalabilnosti itd.

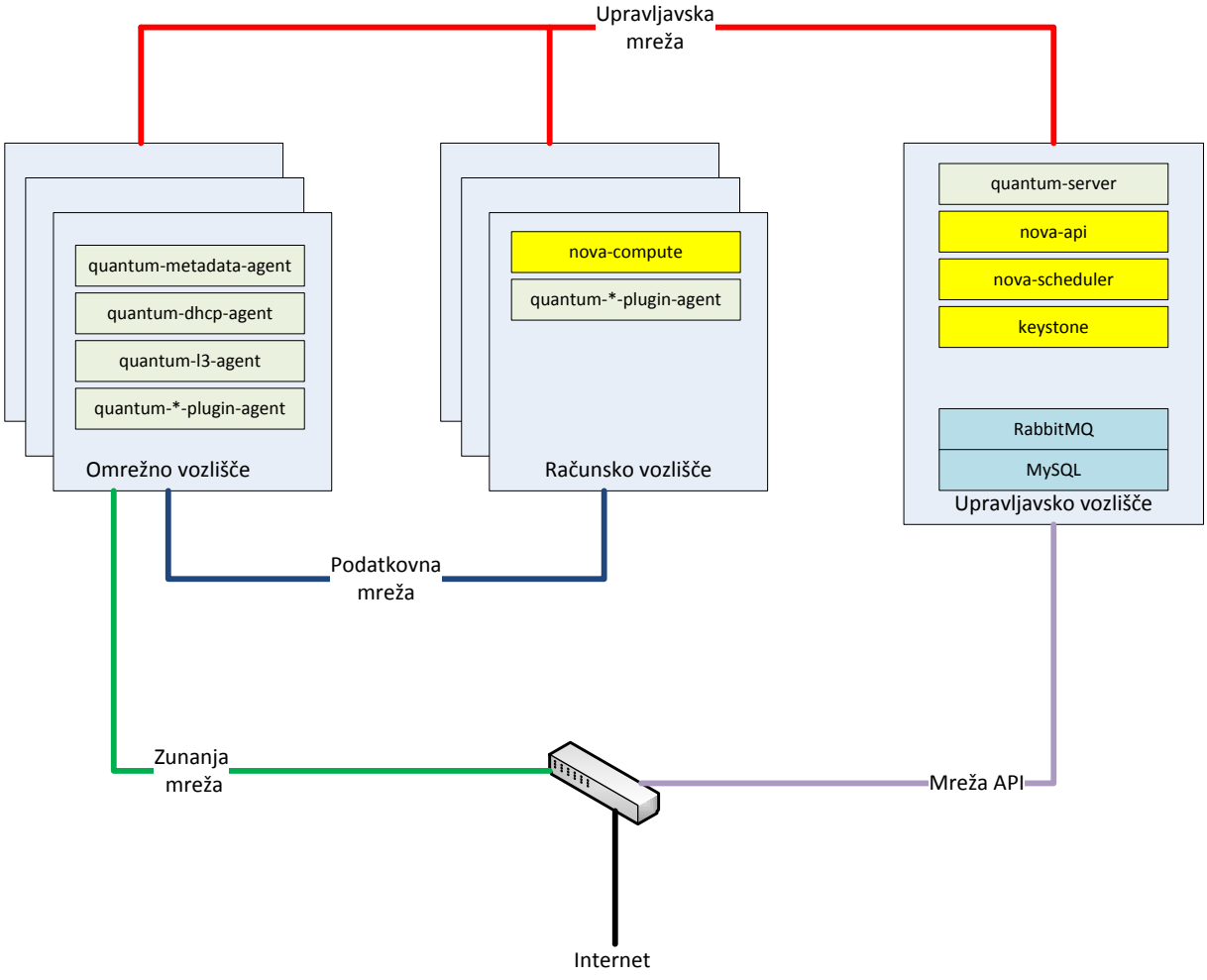
Servis Quantum podobno kot druge servise na platformi OpenStack sestavlja množica med seboj sodelujočih procesov:

- *quantum-server* Demonski proces, napisan v jeziku Python, ki sprejema zahteve prek vmesnika API in jih posreduje naprej konfiguriranemu vtičniku.
- *plugin-agent (quantum-*-plugin-agent)* Proces teče na vsakem računskem vozlišču, kjer opravlja potrebno omrežno konfiguracijo.

- *dhcp-agent (quantum-dhcp-agent)* Zagotavlja storitve DHCP vsem najemniškim omrežjem in je isti ne glede na izbiro vtičnika.
- *l3-agent (quantum-l3-agent)* Zagotavlja NAT-funkcionalnost, ki omogoča virtualnim napravam dostop do zunanjega omrežja.

Ti procesi komunicirajo med sabo prek vmesnika RPC (RabbitMQ) ali prek vmesnika API. Za avtentikacijo in avtorizacijo Quantum uporablja servis Keystone. Tudi spletni vmesnik Dashboard ima vključeno podporo za kreiranje in upravljanje omrežij prek servisa Quantum.

Quantum omogoča precejšnjo fleksibilnost pri namestitvi različnih procesov na računalnike. V primeru uporabe samo enega računalnika lahko vsi procesi tečejo na njem.

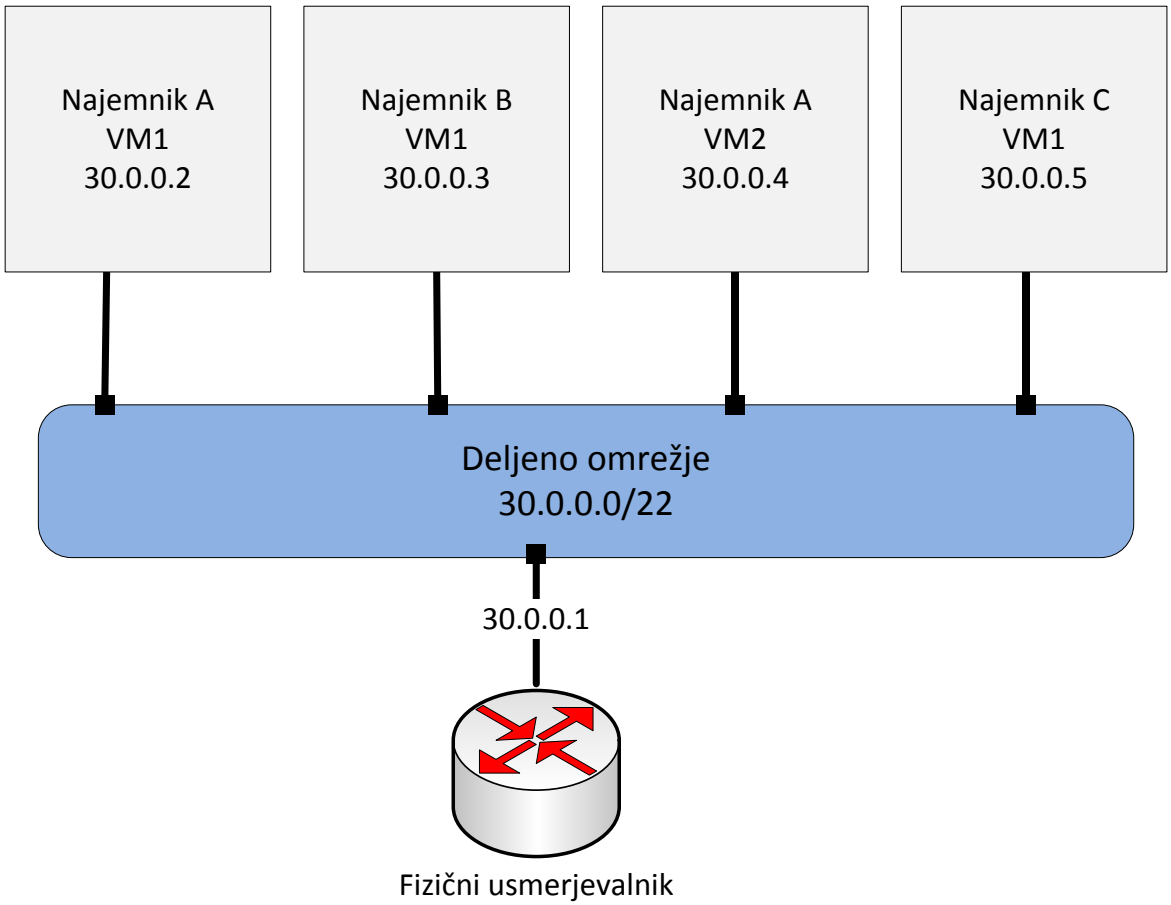


Slika 4.3 Prikaz OpenStack procesov na različnih vozliščih

Slika 4.3 prikazuje standardno arhitekturo namestitve sistema OpenStack. Arhitektura vključuje tri tipe vozlišč: Upravljavsko vozlišče (angl. controller node), enega ali več računskih vozlišč (angl. compute node) in enega ali več omrežnih vozlišč (angl. network node). V določenih primerih je možno omrežna in upravljavska vozlišča združiti. Zgornja namestitev vsebuje štiri fizična omrežja:

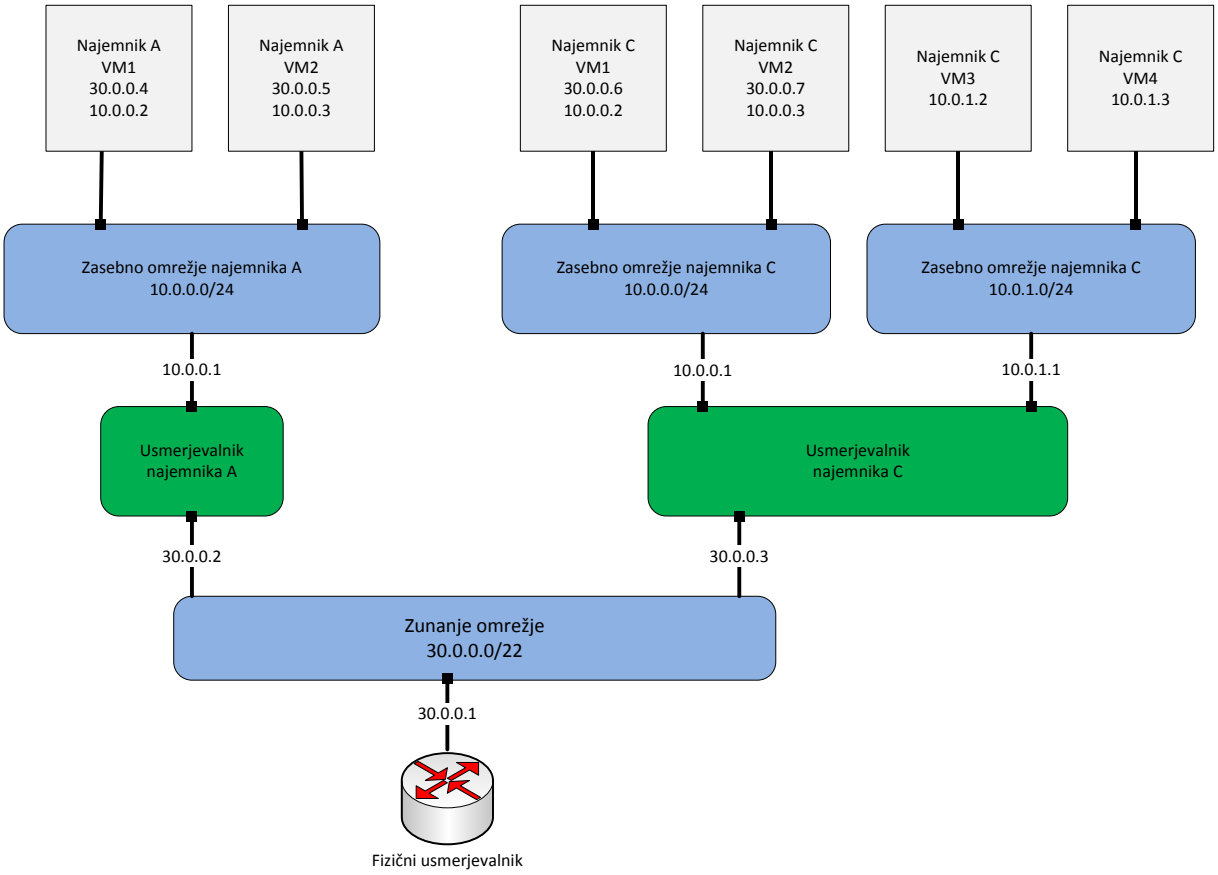
- *Upravljavsko omrežje (angl. management network)* Uporablja se za komunikacijo med različnimi komponentami platforme OpenStack. Naslovi IP morajo biti dosegljivi samo iz samega podatkovnega centra, kjer je oprema nameščena.
- *Podatkovno omrežje (angl. data network)* Uporablja se za komunikacijo med virtualnimi instancami. Tu je treba omeniti, da je možno, da instance uporabnika na enem omrežju tečejo na več računskih vozliščih in morajo prav tako komunicirati prek omrežnega vozlišča.
- *Zunanje omrežje (angl. external network)* Namenjeno je komunikaciji virtualnih instanc s spletom. Naslovi IP na tem omrežju morajo biti zato s spleta dostopni.
- *Omrežje API (angl. API network)* Namenjeno je zunanjemu dostopu najemnikov do vseh vmesnikov API različnih servisov. Naslovi IP na tem omrežju morajo biti dostopni s spleta.

Obstaja nekaj standardnih modelov uporabe servisa Quantum. Ti segajo od najbolj enostavnih pa do precej zapletenih. Tu bomo predstavili najbolj enostaven primer deljenega omrežja in bolj kompleksen primer zasebnih omrežij najemnikov z najemniškimi usmerjevalniki.



Slika 4.4 Enostavno deljeno omrežje za vse najemnike

Na sliki 4.4 je prikazan najenostavnejši primer deljenega omrežja (angl. shared network). Omrežje je vidno vsem najemnikom in ga ustvari administrator sistema. Virtualne instance najemnikov imajo vsaka samo en omrežni vmesnik, ki mu je dodeljen naslov IP z deljenega omrežja. Tako omrežje je po navadi »ponudniško omrežje« (angl. provider network), ki ga administrator kreira kot razširitev že obstoječega omrežja v podatkovnem centru. Instance za dostop do spleta uporabljajo fizični usmerjevalnik.



Slika 4.5 Konfiguracija omrežja z najemniškimi usmerjevalniki

V konfiguraciji na sliki 4.5 najemniki lahko kreirajo svoje usmerjevalnike. Najemniki lahko kreirajo eno ali več zasebnih omrežij. Virtualne instance imajo lahko določene plavajoče naslove IP z zunanjega omrežja, tako da so dostopne od zunaj. Zasebni naslovi IP instanc se lahko med uporabniki prekrivajo, kot je to primer instanc VM1 najemnika A in VM1 najemnika C, ki imata obe zasebni naslov IP 10.0.0.2.

5 Implementacija

V tem poglavju je opisana implementacija rešitve problema hitrega ustvarjanja virtualne infrastrukture, namenjene vaji konfiguriranja strežnika DNS.

Najprej so predstavljene zahteve, ki jih mora opisana rešitev zadovoljiti. Nato je na kratko opisana tehnologija, ki se pri sami izvedbi uporablja. V tretjem delu je bolj natančno opisano ustvarjanje slike operacijskega sistema Ubuntu 12.04.2 LTS, skupaj z namestitvijo in konfiguracijo vseh potrebnih komponent. Sledi del, ki opisuje format šablon Heat ter datotečni format JSON, ki ga le-te uporabljajo. Predstavljena je tudi uporaba projekta Devstack za testiranje implementacije skupaj z vsemi nastavitvami, ki so potrebne za uspešno testiranje. V naslednjem delu so opisane skripte, implementirane v lupini Bash, ki predstavljajo končni uporabniški vmesnik implementacije. Zadnji del opisuje način preverjanja pravilnega delovanja rešitve.

5.1 Zahteve implementacije

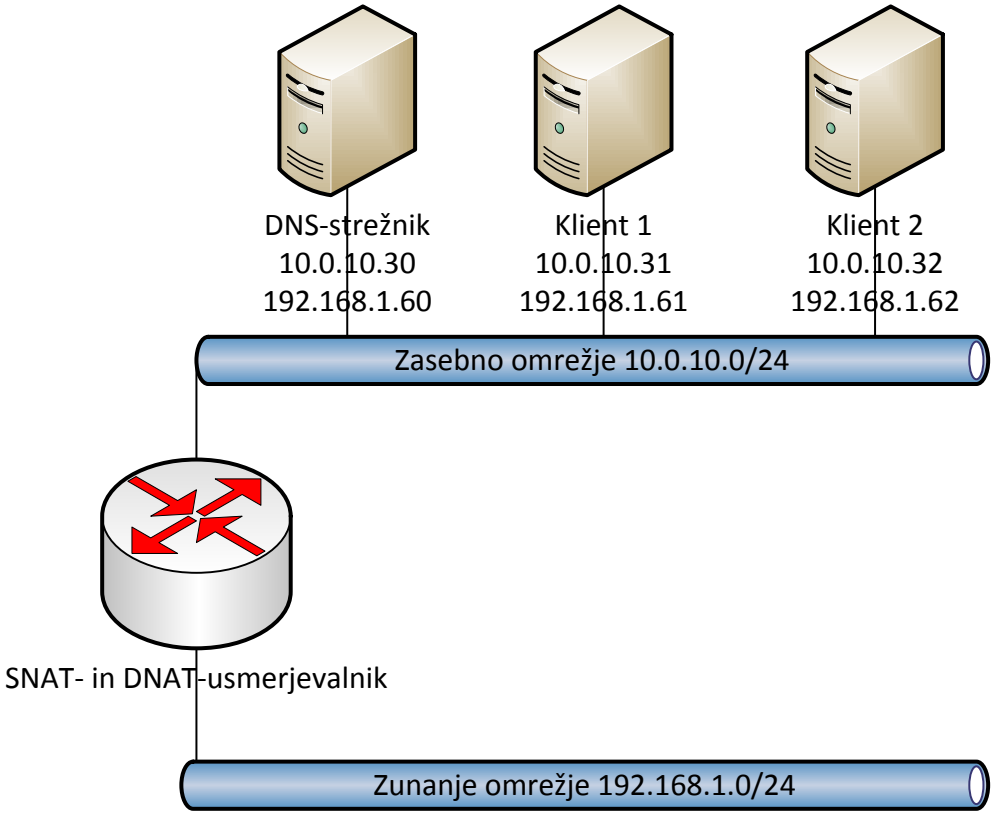
Za implementacijo so pomembne naslednje zahteve:

- *Uporaba platforme OpenStack* V Laboratoriju za računalniške komunikacije je predvidena uporaba platforme OpenStack. OpenStack je vedno bolj popularen. Je odprtokodni projekt, kar pomeni, da je njegova uporaba brezplačna. Z vsako novo izdajo programske opreme je večja stabilnost, čeprav v določenih funkcionalnostih še zaostaja za drugimi podobnimi platformami.
- *Hitro in enostavno kreiranje in brisanje virtualne infrastrukture* Kreiranje in brisanje virtualne infrastrukture uporabnika mora biti zelo enostavno za uporabo in hitro v delovanju.
- *Zunanji omrežni dostop* Virtualne instance uporabnikov morajo biti prek omrežja dostopne z zunanjosti prek protokola SSH.
- *Rešitev mora podpirati prekrivajoče se IP naslove* Ker so naslovi IP instanc uporabnikov enaki, je med njimi treba zagotoviti ločenost. V praksi to pomeni, da imajo instance strežnikov DNS uporabnikov vse enak zasebni naslov IP. Sistem bo

torej deloval pravilno, samo če je zagotovljena izoliranost lokalnih omrežij uporabnikov. Platforma OpenStack tako izoliranost omogoča.

- *Virtualne instance pripravljene na uporabo* Po kreiranju infrastrukture uporabnika, morajo biti virtualne instance pripravljene tako, da sta na instanci strežnika DNS nameščena paketa bind9 in paket dnsutils, na obeh klientih pa samo paket dnsutils.
- *Predpisana topologija virtualne infrastrukture* Virtualna infrastruktura, ustvarjena za enega uporabnika, mora biti enaka strukturi na sliki 5.1. Na zasebnem lokalnem omrežju morajo biti prisotne tri instance. Ena predstavlja strežnik DNS z zasebnim naslovom IP 10.0.10.30, dve pa predstavljata klienta z naslovoma IP 10.0.10.31 ter 10.0.10.32.

  Zasebno omrežje je prek virtualnega usmerjevalnika povezano na omrežje za zunanji dostop. Virtualni usmerjevalnik opravlja izvorno (Source NAT) ter ponorno (Destination NAT) funkcionalnost. Zaradi tega ima vsaka instanca dodeljen še zunanji plavajoči naslov IP. Prek tega naslova IP je instanca dostopna z zunanjega omrežja. Zasebni naslovi IP so pri kreiranju natančno določeni, zunanji plavajoči naslovi IP pa so odvisni od že prej kreiranega zunanjega omrežja, ki je podano kot parameter. To zunanje omrežje že prej ustvari administrator sistema. Kreiranje zunanjega omrežja bo predstavljeno pozneje. Pregled ustvarjene topologije je možen v spletnem vmesniku Dashboard na strani »Network Topology«.



Slika 5.1 Topologija kreirane virtualne infrastrukture enega uporabnika

5.2 Uporabljena tehnologija

Kot osnova rešitve je uporabljena platforma OpenStack. Razlogi so predvsem v hitrem razvoju in velikem obsegu funkcionalnosti, ki se z vsako izdajo veča.

Za hitro orkestracijo uporabnikove infrastrukture je uporabljena komponenta Heat. Heat s pomočjo šablon *(angl. template)* omogoča učinkovit opis konfiguracije. Samodejno kreiranje te konfiguracije je zato hitro in enostavno. Heat omogoča združljivost s platformo Amazon Web Services CloudFormation, zato je možen direkten prenos šablon, ki jih ta uporablja. Heat je za zdaj še inkubacijski projekt, kar pomeni, da še ne ponuja popolne funkcionalnosti in je podprt samo s klientom ukazne vrstice, podpora v spletnem vmesniku Dashboard pa še ni prisotna. V naslednji izdaji OpenStack z imenom Havana je predvidena izdaja Heat v jedru projekta OpenStack.

Kot uporabniški vmesnik so uporabljeni ukazi v obliki skript lupine Bash. Skripte uporabljajo orodja ukazne vrstice komponent Keystone, Heat, Nova ter Quantum.

Za operacijski sistem je uporabljena Linuxova distribucija Ubuntu 12.04 LTS. Ubuntu je zaradi svoje stabilnosti, uporabnosti in prijaznosti do uporabnika zelo primerna izbira. Verzija LTS ima tudi dolg podporni rok, zaradi česar je primeren za uporabo na strežniških sistemih. V repozitorijih programske opreme ima veliko paketov programske opreme, ki jih je možno namestiti.

5.3 Priprava slike operacijskega sistema Ubuntu 12.04 LTS

Slika operacijskega sistema mora izpolnjevati naslednje zahteve: pripravljena mora biti za zagon v virtualnem okolju, nameščeni pa morajo biti paketi cloud-init, heat-cfntools, acpid, openssh-server ter curl.

Za zagon v virtualnem okolju mora biti izpolnjenih več pogojev. V datoteki `/etc/fstab` je treba nastaviti izbiro zagonske particije glede na labelo in dodati labelo zagonski particiji. Pri normalni namestitvi so tu navedene particije z oznako UUID, ki pa se pri uporabi v okolju OpenStack zaradi spreminjanja velikosti diskovnih slik lahko spremenijo. Odstraniti moramo tudi datoteko `/etc/udev/rules.d/70-persistent-net.ruls`. V njej je zabeleženo mapiranje med omrežnimi vmesniki in njihovimi imeni, kot so na primer eth0, eth1 itd. Vsebuje tudi njihove naslove MAC. Datoteko ustvari mehanizem udev ob prvem zagonu sistema. V primeru, ko se konfiguracija in število omrežnih vmesnikov spreminjata, kot je to v primeru zagona v okolju OpenStack, lahko prisotnost te datoteke povzroča težave.

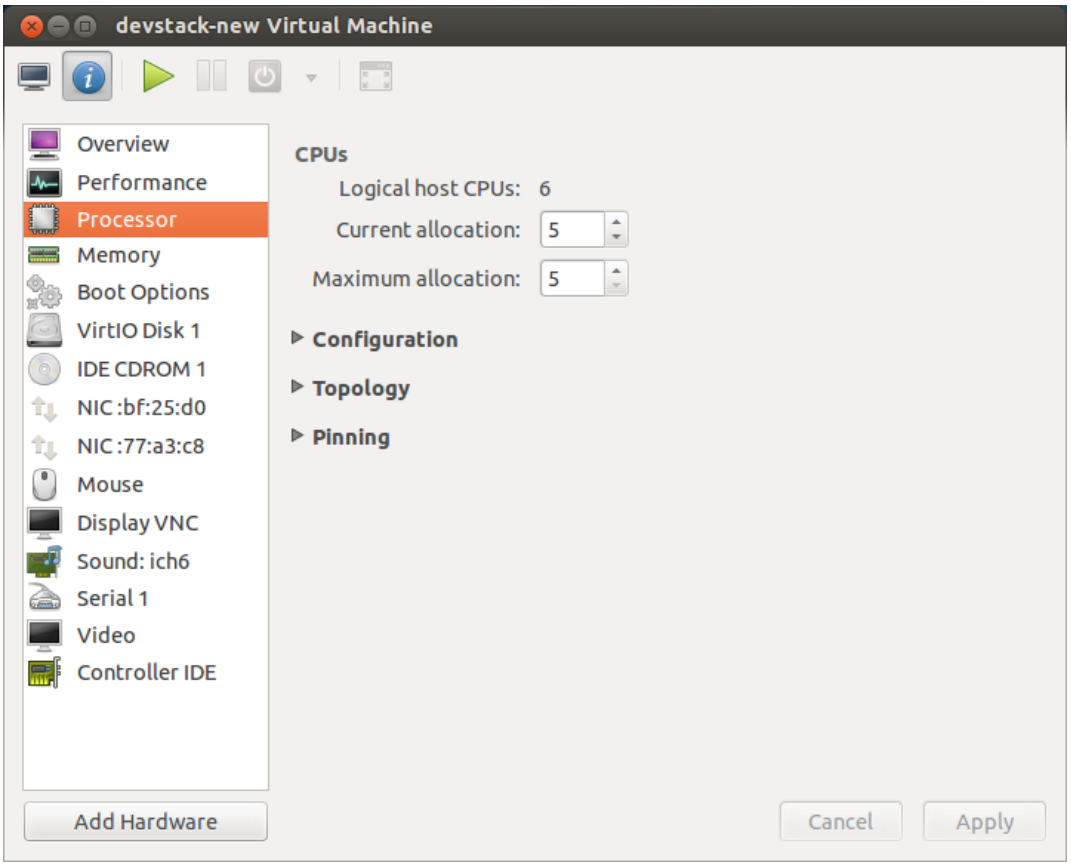
Kreiranje in nameščanje operacijskega sistema je najbolj udobno v programu z grafičnim vmesnikom. Mogoče je tudi iz ukazne vrstice s programom qemu, a je uporaba zaradi velikega števila parametrov otežena. Priporoča se na primer program virt-manager, ki privzeto za virtualizacijo uporablja hipervizor KVM. Vmesnik programa je viden na sliki 5.2. Pomembne nastavitve pri nameščanju so samo velikost in format diskovne datoteke. Zaradi manjše porabe diskovnega prostora se priporoča format QCOW2. Treba je določiti tudi povezavo do omrežja, kjer se lahko izberejo privzete vrednosti. Količina spomina in procesorskih jeder, ki jih določimo, tukaj ni pomembna, ker se te vrednosti ne zapišejo v

samo sliko in bodo ob izvajanju v okolju OpenStack določene posebej. V tej fazi so lahko vrednosti velikodušno izbrane, kar omogoča hitro delovanje in posledično namestitev.

Za namestitev je treba s spleta prenesti namestitveni disk Ubuntu 12.04.2 LTS, ki je na naslovu `http://releases.ubuntu.com/precise/ubuntu-12.04.2-server-amd64.iso`. Ob zagonu namestitve na začetnem zaslonu pritisnemo tipko F4 in izberemo »minimal virtual machine«, s čimer se velikost namestitve zmanjša. Nato lahko namestitev nadaljujemo s privzetimi vrednostmi. Po namestitvi in ponovnem zagonu izvedemo naslednje ukaze:

- `sudo apt-get -y update && sudo apt-get -y upgrade`
- `sudo apt-get -y install curl acpid openssh-server cloud-init`

Ti ukazi posodobijo sistem in namestijo nove programe. S pomočjo programa curl je možno testirati delovanje povezave med OpenStackom in instanco ob njenem zagonu. Demonski proces acpid omogoča nadziranje ponovnega zagona in ugašanja posamezne instance od zunaj.



Slika 5.2 Uporabniški vmesnik virtualizacijskega programa virt-manager.

Openssh-server je strežnik SSH in omogoča povezave SSH do instance. Cloud-init je program, ki se ob zagonu poveže na strežnik z meta podatki in omogoča konfiguracijo sistema na njihovi osnovi. Tudi sistem Heat deluje na osnovi sistema cloud-init. Privzeto cloud-init ne omogoča pridobivanja meta podatkov prek standarda EC2, ki ga uporablja tudi OpenStack. To je treba omogočiti z naslednjim ukazom:

- `sudo dpkg-reconfigure cloud-init`

V izbirniku, ki se pojavi, nato kot vir podatkov izberemo še EC2.

Potrebna je tudi namestitev paketa `heat-cfntools`. Namestitev ne poteka iz repozitorija, ampak se opravi s programom pip:

- `sudo apt-get –y install python-pip`
- `sudo pip install ‘boto==2.5.2’ heat-cfntools`

Prvi ukaz namesti program pip, s katerim nato v drugem ukazu namestimo paket heat-cfntools. V narekovajih je določena tudi namestitev knjižnice boto verzije 2.5.2. Knjižnica boto se uporablja za dostop do virov EC2 in jo uporablja tudi heat-cfntools.

Dodatno je treba konfigurirati še cloud-init. To storimo z nekaj spremembami v datoteki `/etc/cloud/cloud.cfg`. Na vrhu datoteke nastavimo ali dodamo naslednje vrednosti:

- `disable_root: 0`
- `preserve_hostname: False`
- `manage_etc_hosts: True`

Prva možnost omogoči povezavo SSH do instance z uporabnikom root. OpenStack ob zagonu instance v datoteko `/root/.ssh/authorized_keys` injicira javni ključ uporabnika. Uporabnik se lahko nato s svojim zasebnim ključem preko odjemalca SSH poveže z instanco kot uporabnik root. Če ima nastavitev vrednost 1, ta povezava ni mogoča. Drugi dve nastavitvi omogočata spreminjanje datotek `/etc/hosts` in `/etc/hostname` in s tem spreminjanje imena instance prek sistema cloud-init.

Nadalje naredimo še zgoraj omenjeno spremembo datoteke /etc/fstab. V ta namen na sistemu Ubuntu modificiramo diskovno sliko, ki smo jo pred tem ustvarili. Za to uporabimo program qemu-nbd:

- `modprobe nbd max_part=63`
- `qemu-nbd –c /dev/nbd0 image.img`

Sedaj se slika nahaja na napravi `/dev/nbd0`. S pomočjo programa fdisk lahko izpišemo particije v tej sliki. Na zagonsko particijo damo labelo, ki bo omogočala zagon prek labele. Če je v sliki samo ena particija, ima ime `/dev/nbd0p1` in je seveda tudi zagonska. Trenutno labelo izpišemo z ukazom:

- `sudo e2label /dev/nbd0p1`

Labelo spremenimo z ukazom:

- `sudo e2label /dev/nbd0p1 ubuntu-root`

S prejšnjim ukazom preverimo, ali je bila labela res spremenjena.

Labelo je treba nastaviti še v datoteki `/etc/fstab`. To naredimo tako, da v vrstici particije zamenjamo UUID=..... z LABEL=ubuntu-root.

5.4 Implementacija šablone Heat

Šablone Heat so napisane v tekstovnem formatu JSON (JavaScript Object Notation). Heat uporablja šablone, ki se uporabljajo tudi v Amazonovem okolju Amazon Web Services. Trenutno še ni zagotovljena popolna združljivost, saj precej virov, ki jih Amazon uporablja, še ni podprtih. Razvoj pa poteka precej hitro in tako se tudi združljivost hitro veča.

5.4.1 Tekstovni format JSON

JSON je odprt standard tekstovnih datotek, razvit za potrebe izmenjave podatkov v človeško berljivi obliki. Omogoča hitro strojno razčlenjevanje in generiranje. Ustvarjen je kot podmnožica programskega jezika JavaScript. Kljub temu se enostavno uporablja tudi v drugih jezikih, kot so C, C++, Python [6]. Format JSON je podrobno opisan v [7]. Uporaben je tudi kot alternativa jeziku XML.

Osnovni tipi v formatu JSON so:

- *Števila (angl. numbers)* Števila dvojne natančnosti v formatu plavajoče vejice.
- *Niz (angl. string)* Nizi v formatu Unicode.
- *Logične vrednosti (angl. boolean)* Vrednosti »da« in »ne« (angl. »true« in »false«).
- *Matrika (angl. array)* Urejen niz vrednosti, ločenih z vejico, ki so med oglatimi oklepaji. Vrednosti so lahko različnih tipov.
- *Objekt (angl. object)* Neurejena zbirka parov ključ:vrednost, kjer sta ključ in vrednost ločena z dvopičjem. Pari so ločeni z vejico. Objekt je omejen z zavitima oklepajem in zaklepajem.

Objekt ali matrika seveda lahko vsebujeta druge objekte ali matrike.

Primer objekta JSON [7]:

```
{
    "Image": {
        "Width":  800,
        "Height": 600,
        "Title":  "View from 15th Floor",
        "Thumbnail": {
            "Url":    "http://www.example.com/image/481989943",
            "Height": 125,
            "Width":  "100"
        },
        "IDs": [116, 943, 234, 38793]
    }
}
```

5.4.2 Format šablon Heat

Heat uporablja enako strukturo šablone kot storitev podjetja Amazon, Amazon Web Services Cloud Formation. V OpenStack Heat so implementirani še dodatni viri, ki jih AWS ne uporablja. Taki viri so na primer različne funkcionalne enote iz omrežnega modula Quantum, kot so omrežja, podomrežja, usmerjevalniki, vmesniki, plavajoči naslovi IP itd.

Šablone Heat morajo biti napisane v formatu JSON. Šablona omogoča kreiranje in brisanje skupine virov kot enote. Ta enota se imenuje sklad *(angl. stack).* V njej so definirani parametri *(angl. parameters),* mapiranja *(angl. mappings),* značilnosti virov *(angl. resource properties)* in izhodne vrednosti *(angl. output values)* [8].

Sklad je zbirka različnih virov. S pomočjo ukaza `heat stack-create` uporabnik lahko ustvari sklad, z ukazom `heat stack-delete` ga lahko izbriše, z ukazom `heat event-list` pa izpiše dogodke v času ustvarjanja ali brisanja.

Šablona je objekt JSON, ki lahko vsebuje šest drugih objektov: verzija formata *(angl. format version),* opis *(angl. description),* parametri *(angl. parameters),* mapiranje *(angl. mappings),* viri *(angl. resources)* in izhodne vrednosti *(angl. outputs).* Vsi ti glavni objekti, razen virov, so neobvezni.

Primer enostavne veljavne šablone:

```
{
    "Resources" : {
        "MyQueue" : {
            "Type" : "AWS::SQS::Queue",
            "Properties" : {
            }
        }
    }
}
```

Ta šablona vsebuje samo vir. Bolj kompleksna šablona, ki je bila razvita v okviru tega dela, je v prilogi 7.1.

Objekt verzija formata določa verzijo formata, uporabljeno v tej šabloni. Opis vsebuje tekstovni niz, ki opisuje funkcionalnost šablone.

Parametri so vrednosti v šabloni, definirani v sekciji parametrov. Parametri imajo lahko privzete vrednosti in so lahko različnih tipov: niz *(angl. String),* število *(angl. Number),* seznam *(angl. CommaDelimitedList).* Možno je določiti omejitve parametrov, kot so minimalne in maksimalne vrednosti ter minimalna ter maksimalna dolžina. Primer parametra:

```
"LinuxDistribution": {
      "Default": "U12",
```

```
        "Description" : "Distribution of choice",
        "Type": "String",
        "AllowedValues" : "U12"
  }
```

Mapiranja omogočajo specifikacijo pogojnih parametrov. S funkcijo `Fn::FindInMap` potem lahko mapiranje uporabimo v maniri case stavka. Primer mapiranja:

```
"Mappings" : {
    "AWSInstanceType2Arch" : {
      "t1.micro"    : { "Arch" : "32" },
      "m1.small"    : { "Arch" : "32" },
      "m1.large"    : { "Arch" : "64" },
      "m1.xlarge"   : { "Arch" : "64" },
      "m2.xlarge"   : { "Arch" : "64" },
      "m2.2xlarge"  : { "Arch" : "64" },
      "m2.4xlarge"  : { "Arch" : "64" },
      "c1.medium"   : { "Arch" : "32" },
      "c1.xlarge"   : { "Arch" : "64" },
      "cc1.4xlarge" : { "Arch" : "64" },
      "m1.micro"    : { "Arch" : "64" }
    },
    "DistroArch2AMI": {
      "U12"      : { "32" : "U12-i386-cfntools", "64" : "U12-x86_64-cfntools" }
    }
  }
```

V vsaki šabloni mora biti prisoten vsaj en vir. Vsak vir ima določeno ime in tip vira. Vsak vir ima sekcijo značilnosti *(angl. properties)*, kjer so določene obvezne ali opcijske značilnosti. Primer vira:

```
"subnet": {
      "Type": "OS::Quantum::Subnet",
      "Properties": {
        "network_id": { "Ref" : "network" },
        "ip_version": 4,
        "cidr": "10.0.10.0/24",
        "allocation_pools": [{"start": "10.0.10.20", "end": "10.0.10.50"}]
      }
},
```

Ta sekcija deklarira vir. Kot tip je deklarirano Quantum podomrežje. Deklarirana je verzija IP 4, CIDR-oznaka omrežja ter množica naslovov IP, ki jih lahko uporabljajo naprave, priključene nanjo. S funkcijo `Ref`, ki se uporablja za vrednost `network_id`, se sklicujemo oziroma delamo povezave do drugih virov.

Šablone lahko vsebujejo tudi vdelane funkcije. Trenutno so podprte naslednje:

- *Fn::Base64* Vrne argument kodiran v obliki base64.
- *Fn::FindInMap* Vrne vrednost ključa iz specificiranega mapiranja.
- *Fn::GetAtt* Vrne vrednost atributa specificiranega vira.
- *Fn::GetAZs* Vrne dovoljene zone *(angl. availability zones)*, kjer je dovoljeno kreirati sklad.
- *Fn::Join* Združevanje elementov v drugem argumentu, ločenih s prvim argumentom.
- *Ref* Vrne vir ali vrednost v odvisnosti od logičnega imena ali parametra.

5.4.3 Opis implementacije šablone

V tem razdelku sledi opis šablone, ki je bila kreirana v okviru tega dela. Šablona se nahaja v prilogi 7.1.

V prvih dveh sekcijah se nahajata verzija formata ter opis šablone. Sledijo parametri `KeyName`, `ExtNetUuid`, `InstanceType` in `LinuxDistribution`. Parameter KeyName določa ključ, ki se uporabi pri ustvarjanju instanc in omogoča povezavo SSH do nje. Parameter ExtNetUuid določa podano Quantum omrežje, ki se uporabi za povezavo instanc do zunanjega omrežja. Ta dva parametra morata biti obvezno podana. Parameter InstanceType določa tip instance, ki bo uporabljen pri njenem kreiranju. Privzeta vrednost je m1.micro. Parameter LinuxDistribution določa sliko operacijskega sistema, ki se uporabi pri kreiranju instanc. Tudi ta parameter ima privzeto vrednost U12, kar pomeni, da se pri kreiranju instance uporabi Ubuntu 12.04.

Sledita dve mapiranji, in sicer mapiranje iz tipa instance v arhitekturo in iz imena distribucije v ime slike operacijskega sistema.

Sledi sekcija z navedenimi viri. Prvi vir je Quantum omrežje tipa OS::Quantum::net z imenom network. Nanjo se nanaša tudi drugi vir, Quantum podomrežje tipa OS::Quantum::subnet z imenom subnet. Sledijo trije viri DNSServerPort, Client1Port in Client2Port tipa OS::Quantum::Port. Ti viri so uporabljeni kot omrežni vmesniki instanc, ki se kreirajo v šabloni. Razlog, da morajo biti specificirani posebej, je v tem, da je le na ta način možno določiti naslove IP omrežnih vmesnikov. Sledi definicija virov router tipa OS::Quantum::router, router_interface_private tipa OS::Quantum::RouterInterfac in router_gateway_external tipa OS::Quantum::RouterGateway. Vir router predstavlja enostaven statični usmerjevalnik, ki je prek obeh vmesnikov povezan na zasebno in zunanje omrežje. Vmesnik router_gateway_external opravlja funkcionalnost NAT. Sledita dve varnostni grupi DNSServerSecurityGroup in MinimalSecurityGroup tipa AWS::EC2::SecurityGroup. Varnostne grupe definirajo dovoljen vhodni promet za instance. Za strežnik DNS je dovoljen ICMP (ping), SSH in DNS (vrata 53) promet. Za kliente pa samo ICMP in SSH promet. Sledijo viri instanc DNSServer, Client1 in Client2 tipa AWS::EC2::Instance. V sekciji Metadata imajo določene pakete, ki naj se ob zagonu namestijo nanje. Določene imajo tudi druge značilnosti, kot so uporabljeni ključ, tip instance, varnostne grupe in omrežni vmesnik. V značilnosti UserData pa je specificirana skripta, ki se izvede ob zagonu instance. Ta nastavi strežnik DNS in zažene skripto cfn-init, ki sproži zagon konfiguracije, kot je na primer namestitev specificiranih programskih paketov. Sledijo viri tipov OS::Quantum::FloatingIP in OS::Quantum::FloatingIPAssociation, ki instancam dodelijo plavajoče naslove IP z zunanjega omrežja, kar omogoči zunanji IP dostop do njih.

Sledi sekcija z izhodnimi vrednostmi, ki pa je v tem primeru prazna.

5.5 Priprava testnega okolja z uporabo projekta Devstack

Za čim enostavnejše testiranje funkcionalnosti je priporočljiva uporaba projekta Devstack. Projekt je namenjen hitri namestitvi okolja OpenStack v testne ali razvojne namene. Po navadi se okolje namesti na en sam računalnik, mogoča pa je tudi porazdeljena namestitev na več računalnikov. V okviru razvoja tega dela je bila testirana uporaba na operacijskem

sistemu Ubuntu 12.04 LTS. Namestitev je možna direktno na osnovni operacijski sistem, možna pa je tudi uporaba virtualizacije. V primeru uporabe le-te se priporoča uporaba virtualizacijske tehnologije KVM, ker ta omogoča ugnezdeno virtualizacijo. Za testiranje vse funkcionalnosti mora računalnik oziroma virtualni računalnik imeti dva omrežna vmesnika. Prvi vmesnik je namenjen splošni povezljivosti in upravljanju sistema, drugi pa je namenjen dostopu virtualnih instanc do zunanjega omrežja.

V prvem koraku izvedemo standardno namestitev operacijskega sistema Ubuntu 12.04 LTS.

Drugi korak je namestitev okolja Devstack. V ta namen je treba najprej klonirati repozitorij projekta Devstack:

- `git clone git://github.com/openstack-dev/devstack.git`

Devstack uporablja za konfiguriranje datoteko `localrc`. Datoteko damo v korenski direktorij kloniranega Devstack repozitorija. Predlagana vsebina datoteke je prikazana v prilogi 7.6. Datoteka nastavi gesla za različne servise v okolju OpenStack, onemogoči servis nova-network in omogoči omrežni servis Quantum. Omogoči tudi servis Heat in kot primer doda sliko operacijskega sistema Fedora. V okviru teh navodil bomo nato ročno dodali sliko sistema Ubuntu, ki smo jo ustvarili v razdelku 5.3.

Namestitev sistema OpenStack nato sprožimo z izvedbo ukaza:

- `./stack.sh`

Namestitev poteka nekaj minut. V primeru ponovnega zagona računalniškega sistema ni potrebna ponovna izvedba zgornjega ukaza, ampak se izvede ukaz `./rejoin-stack.sh`.

V okviru namestitve Devstack je treba omogočiti tudi funkcionalnost meta podatkov, ki omogoči komunikacijo instanc s sistemom OpenStack, kar je potrebno za njihovo modifikacijo. Prek tega mehanizma, denimo, servis Heat namesti programe, ki so navedeni v šabloni Heat. Funkcionalnost meta podatkov omogočimo tako, da nastavimo naslednje vrednosti v konfiguracijskih datotekah:

```
/etc/nova/nova.conf

  quantum_metadata_proxy_shared_secret=password
  service_quantum_metadata_proxy=true
```

```
  metadata_listen = 10.10.10.10
  metadata_listen_port = 8775
```

/etc/quantum/dhcp_agent.ini

```
  [DEFAULT]
  enable_isolated_metadata = True
  enable_metadata_network = True
```

/etc/quantum/metadata_agent.ini

```
  [DEFAULT]
  auth_url = http://10.10.10.10:35357/v2.0
  auth_region = RegionOne
  admin_tenant_name = service
  admin_user = quantum
  admin_password = password
  nova_metadata_ip = 10.10.10.10
  metadata_proxy_shared_secret = password
```

Dodatno je treba povečati dovoljeno hitrost sprejemanja ukazov API, ker drugače skripte lahko sprožijo hitrostno omejevanje. To storimo z naslednjo spremembo:

/etc/nova/api-paste.ini

```
  [filter:ratelimit]
  paste.filter_factory =
nova.api.openstack.compute.limits:RateLimitingMiddleware.factory
  limits =(POST, "*", .*, 20, MINUTE);(POST, "*/servers", ^/servers, 50, DAY);(PUT,
"*", .*, 10, MINUTE);(GET, "*changes-since*", .*changes-since.*, 3, MINUTE);(DELETE,
"*", .*, 100, MINUTE)
```

Dodati je treba samo tretjo vrstico, kjer je definirana spremenljivka limits. Spremenjena je samo prva vrednost z 10 na 20, preostale vrednosti so enake privzetim.

Po teh nastavitvah je potreben ponovni zagon sistema in izvedba ukaza `./rejoin-stack.sh`, ki zažene OpenStack. Naslov IP 10.10.10.10 je naključen, navesti moramo pravilen naslov IP sistema, kjer poteka namestitev.

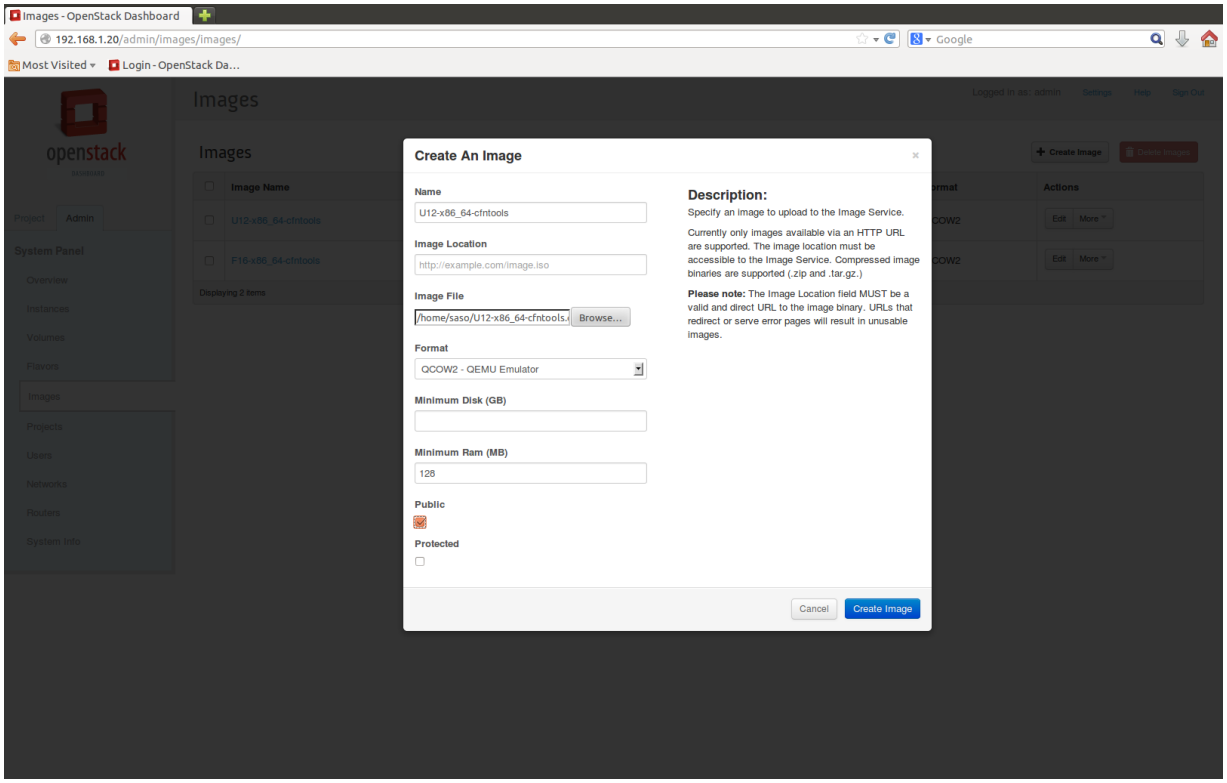
Delovanje funkcionalnosti meta podatkov lahko preverimo z naslednjima ukazoma:

```
root@jvzorec-dns-ex:~# curl http://169.254.169.254/2009-04-04/meta-data

ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
kernel-id
local-hostname
local-ipv4
placement/
public-hostname
public-ipv4
public-keys/
ramdisk-id
reservation-id
security-groupsroot@jvzorec-dns-ex:~#

root@jvzorec-dns-ex:~# curl http://169.254.169.254/2009-04-04/user-data

Content-Type: multipart/mixed; boundary="===============4159756969596254141=="
MIME-Version: 1.0

--===============4159756969596254141==
Content-Type: text/cloud-config; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="cloud-config"

user: ec2-user

cloud_config_modules:
 - locale
 - set_hostname
 - timezone
 - update_etc_hosts
 - update_hostname

...
```

```
[Boto]
debug = 0
is_secure = 0
https_validate_certificates = 1
cfn_region_name = heat
cfn_region_endpoint = 192.168.1.20
cloudwatch_region_name = heat
cloudwatch_region_endpoint = 192.168.1.20
--===============4159756969596254141==--root@jvzorec-dns-ex:~#
```

S tema ukazoma izpišemo meta-data in user-data podatke. Izpis user-data je skrajšan, ker vsebuje veliko količino besedila. V teh podatkih je prisotna tudi koda Python, ki jo generira sistem Heat in ki poskrbi za namestitev programov v instanco.

Sliko sistema Ubuntu dodamo v spletnem vmesniku Dashboard na strani Images kot administratorski uporabnik.



Slika 5.3 Dodajanje slike sistema v vmesniku Dashboard

Poskrbeti je treba, da je sliki določeno ime U12-x86_64-cfntools. To ime je določeno tudi v šabloni Heat. Sliko označimo kot javno *(angl. public)*, da jo lahko uporabljajo vsi uporabniki. Določita se lahko tudi minimalni količini spomina in diska, ki sta potrebni za

njeno pravilno delovanje. Sliko je mogoče dodati tudi iz ukazne vrstice s pomočjo ukaza `glance image-create`.

Nadalje je treba kreirati zunanje omrežje, ki je podano kot parameter v procesu kreiranja sklada. Postopek:

```
$ quantum net-create --router:external=True ext_net
$ quantum subnet-create --name ext_subnet --ip-version 4 –gateway 192.168.1.100
--allocation-pool start=192.168.1.60,end=192.168.1.70 --disable-dhcp ext_net
192.168.1.0/24
$ quantum net-update --shared=True ext_net
```

V prvem koraku ustvarimo omrežje in s parametrom `–router:external=True` določimo, da se na njem omogoči funkcionalnost NAT. Drugi korak kreira na omrežju podomrežje, določi verzijo IP 4, določi prehodni strežnik *(angl. gateway)* in določi obseg plavajočih naslovov IP, ki bodo dodeljeni instancam. Izklopi se tudi dhcp funkcionalnost na omrežju, ki tukaj ni potrebna. Prikazane nastavitve služijo samo kot primer, naslovi IP morajo biti prilagojeni za uporabo v določenem okolju.

Privzeto se za implementacijo Quantum omrežij uporablja vtičnik Open vSwitch. Za dostop instance do zunanjega omrežja virtualni most `br-ex` povežemo z omrežnim vmesnikom, ki se uporablja za zunanji dostop. Po navadi je ta vmesnik eth1. Izvedemo ukaz:

```
$ sudo ovs-vsctl add-port eth1 br-ex
```

Potem ko so izvedeni zgornji koraki, bi moral biti sistem pripravljen za uporabo skript, ki so opisane v naslednjem delu.

5.6 Opis skript

Za kreiranje in brisanje uporabniške virtualne infrastrukture so bile razvite štiri skripte v lupini Bash:

- *ostack_create_tenant_user_dns-exercise.sh*
- *ostack_stack_list_dns-exercise.sh*
- *ostack_purge_tenant_user_dns-exercise.sh*

- *ostack_purge_all_tenants_dns-exercise.sh*

Prva skripta se uporablja za kreiranje uporabnikove infrastrukture, druga izpiše stanje skladov uporabnikov, tretja izbriše infrastrukturo enega uporabnika in četrta izbriše infrastrukturo vseh uporabnikov. Za uspešno izvajanje skript nastavimo okoljske spremenljivke. Skripte morajo biti izvedene v imenu administratorskega uporabnika. V ta namen je možno ročno ustvariti datoteko s potrebnimi nastavitvami, bolj enostavno pa je datoteko prenesti iz spletnega vmesnika Dashboard. To naredimo na strani Access & Security, zavihek API Access, gumb »Download Openstack RC file«. Izgled datoteke je naslednji:

```
#!/bin/bash

# With the addition of Keystone, to use an openstack cloud you should
# authenticate against keystone, which returns a **Token** and **Service
# Catalog**.  The catalog contains the endpoint for all services the
# user/tenant has access to - including nova, glance, keystone, swift.
#
# *NOTE*: Using the 2.0 *auth api* does not mean that compute api is 2.0.  We
# will use the 1.1 *compute api*
export OS_AUTH_URL=http://10.0.100.102:5000/v2.0

# With the addition of Keystone we have standardized on the term **tenant**
# as the entity that owns the resources.
export OS_TENANT_ID=feacce5a1fc347f88cfc0dee838429d6
export OS_TENANT_NAME=tenant

# In addition to the owning entity (tenant), openstack stores the entity
# performing the action as the **user**.
export OS_USERNAME=username

# With Keystone you pass the keystone password.
echo "Please enter your OpenStack Password: "
read -s OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT
```

Datoteko pred izvedbo skript izvedemo z ukazom:

```
$ source admin-openrc.sh
```

Ukaz nas pozove na vpis gesla. Sledi opis uporabe skript.

5.6.1 ostack_create_tenant_user_dns-exercise.sh

Skripta kreira virtualno infrastrukturo s podanim imenom. Prav tako je podana oznaka zunanjega omrežja, ki se uporabi za povezljivost instanc. Skripta je prikazana v prilogi 7.2. Pred uporabo z ukazom quantum net-list pogledamo, katero omrežje moramo podati kot parameter:

```
saso@devstack-new-virt:~$ quantum net-list
+--------------------------------------+---------+--------------------------------------------------
----+
|  id                                                                  |  name        |  subnets
|
+--------------------------------------+---------+--------------------------------------------------
----+
|   3c130c14-6dad-4fc8-9927-0fe2de4e3c1d   |   ext_net   |   02538e8d-530a-4117-939e-10491c55b0fa
192.168.1.0/24 |
+--------------------------------------+---------+--------------------------------------------------
----+
saso@devstack-new-virt:~$
```

Podati je treba parameter pod oznako id.

Primer uporabe:

```
saso@devstack-new-virt:~/diploma-skripte$
./ostack_create_tenant_user_dns-exercise.sh jvzorec
34fddfd3-6fe5-4c13-845b-ddbc3fec49ed
Created user and tenant: jvzorec-DNS-EX
Password: jvzorec-DNS-EX
Private key stored into: jvzorec-DNS-EX-key.pem
+--------------------------------------+----------------+--------------------+----------
------------+
| id                                   | stack_name     | stack_status       |
creation_time        |
+--------------------------------------+----------------+--------------------+----------
------------+
| 09a2a50b-8fa3-4a0d-8c1e-ba2f8d362c0b | jvzorec-DNS-EX | CREATE_IN_PROGRESS |
2013-05-15T10:55:15Z |
+--------------------------------------+----------------+--------------------+----------
------------+
Created DNS stack: jvzorec-DNS-EX

saso@devstack-new-virt:~/diploma-skripte$
```

Skripta najprej preveri obstoj omrežja, ki je podano kot parameter. Iz podanega uporabniškega imena kreira novo, tako da doda končnico `–DNS-EX`. Razlog leži v tem, da se tako ustvarjena imena najemnikov in uporabnikov lažje ločijo od imen sistemskih uporabnikov in najemnikov, kar olajša njihov izpis in brisanje. Nadalje skripta ustvari najemnika in uporabnika. Kreira tudi javni in zasebni ključ, ki se uporablja za dostop SSH do instanc. Nato uporabi Heat za samo kreiranje sklada in izpiše njegov status. V tem trenutku lahko novi uporabnik že izvede prijavo v sistem in opazuje kreiranje njegove infrastrukture. Geslo je enako imenu uporabnika in najemnika.

5.6.2 ostack_stack_list_dns-exercise.sh

Skripta je namenjena izpisu vseh ustvarjenih skladov v sistemu in je prikazana v prilogi 7.3. Poišče vse najemnike, ki imajo v imenu prisoten tekst `–DNS-EX`, in izpiše njihove sklade. Primer uporabe:

```
   saso@devstack-new-virt:~/diploma-skripte$ ./ostack_stack_list_dns-exercise.sh
   TENANT: jvzorec-DNS-EX
   +--------------------------------------+---------------+-----------------+------------
---------+
   | id                                   | stack_name     | stack_status    |
   creation_time        |
   +--------------------------------------+---------------+-----------------+------------
---------+
   | 09a2a50b-8fa3-4a0d-8c1e-ba2f8d362c0b | jvzorec-DNS-EX | CREATE_COMPLETE |
   2013-05-15T10:55:15Z |
   +--------------------------------------+---------------+-----------------+------------
---------+
   saso@devstack-new-virt:~/diploma-skripte$
```

Tu so izpisani podatki za samo enega najemnika, ker je trenutno ustvarjena samo njegova infrastruktura. Če je prisotnih najemnikov več, se izpišejo podatki za vse.

5.6.3 ostack_purge_tenant_user_dns-exercise.sh

Skripta je namenjen brisanju uporabnikove infrastrukture. Prikazana je v prilogi 7.4. Skripta sproži brisanje sklada prek sistema Heat. Nato čaka, da je sklad uspešno odstranjen in izbriše najemnika in uporabnika. Čakanje je potrebno zato, ker v primeru brisanja najemnika

in uporabnika, preden je sklad zbrisan, sklad ni popolnoma odstranjen iz sistema. To skripto uporablja tudi naslednja skripta, ki iz sistema odstrani vse najemnike in njihovo infrastrukturo.

5.6.4 ostack_purge_all_tenants_dns-exercise.sh

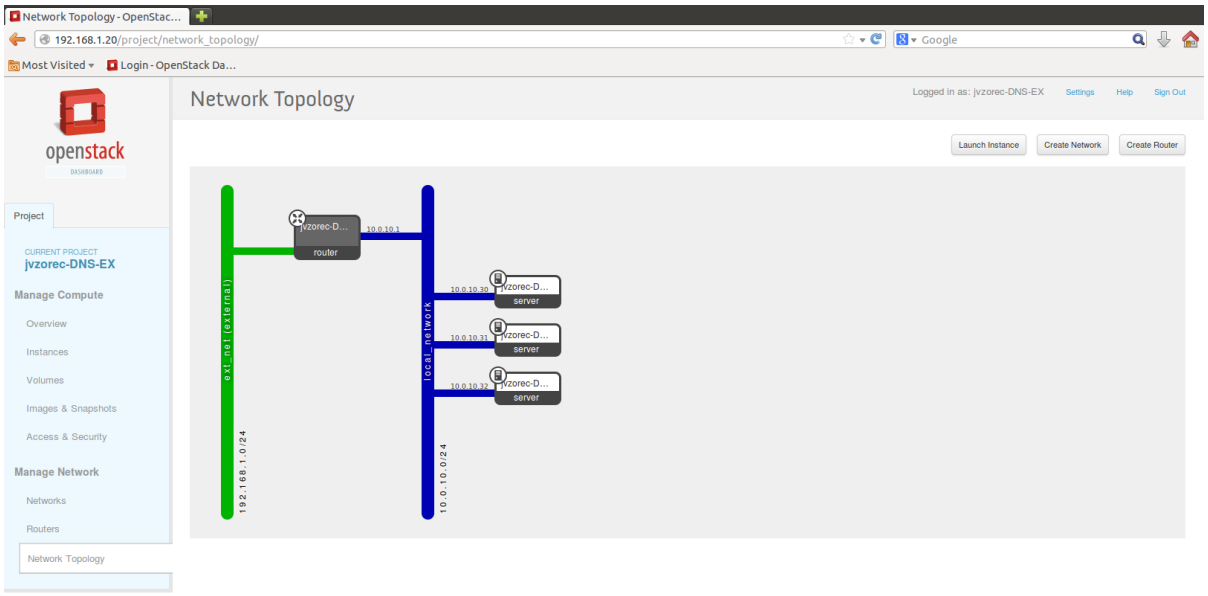
Skripta odstrani vse najemnike in njihovo infrastrukturo s sistema. Prikazana je v prilogi 7.5. Primer uporabe:

```
saso@devstack-new-virt:~/diploma-skripte$
./ostack_purge_all_tenants_dns-exercise.sh
Purging the following tenants in the background
jvzorec-DNS-EX

saso@devstack-new-virt:~/diploma-skripte$
```

Skripta izpiše imena vseh najemnikov, ki bodo brisani. Brisanje poteka v ozadju in lahko traja še nekaj minut po zagonu ukaza. Iskanje najemnikov poteka po vsebovanosti teksta DNS-EX.

5.7 Preverjanje pravilnosti delovanja

V tem delu je opisano preverjanje pravilnosti delovanja funkcionalnosti. Po kreiranju infrastrukture uporabnika moramo najprej preveriti, ali se uporabnik lahko uspešno prijavi v sistem prek spletnega vmesnika Dashboard. Tu moramo upoštevati, da je geslo enako imenu uporabnika. V vmesniku Dashboard se najprej preveri, ali topologija infrastrukture ustreza naslednji sliki:



Slika 5.4 Topologija pravilno ustvarjene infrastrukture uporabnika

Izgled mora biti enak kot na sliki 5.4. Ustvarjene morajo biti tri instance na zasebnem omrežju, ki je prek virtualnega usmerjevalnika povezana z zunanjim omrežjem. Naslovi IP instanc morajo biti 10.0.10.30, 10.0.10.31 in 10.0.10.32.

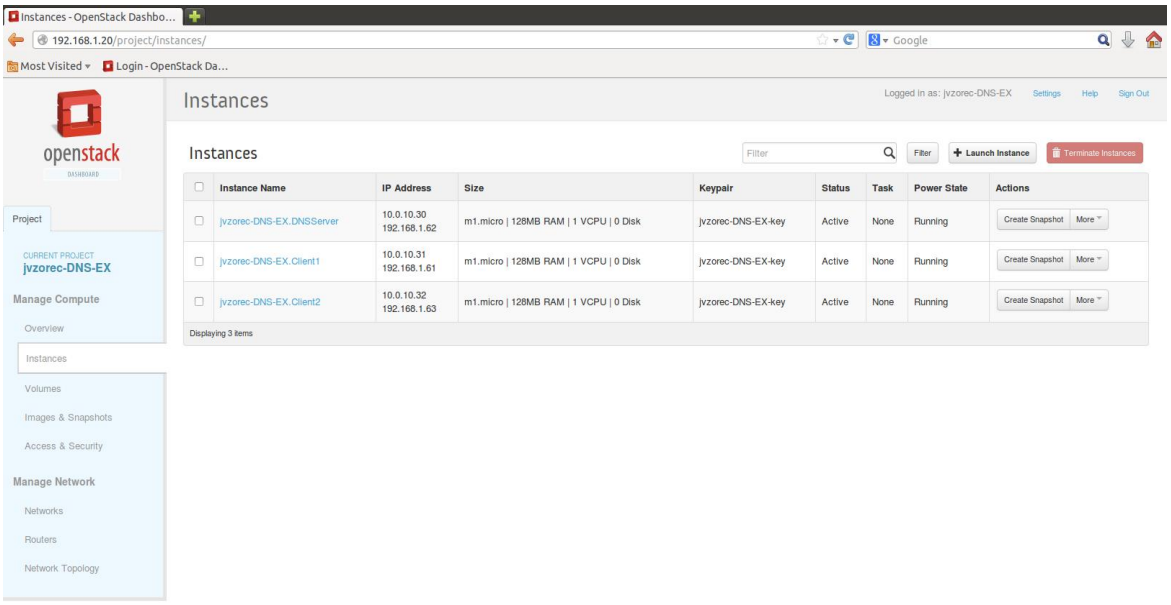
Na strani pregleda ustvarjenih instanc morajo biti izpisane vse tri. Izpisani morajo biti njihovi zasebni in plavajoči naslovi IP, kot je to prikazano na sliki 5.5.

Izvesti moramo še preverjanje odzivnosti na ukaz ping. V ta namen izvedemo ukaz ping, ki ima za parameter plavajoči naslov IP ene od instanc:

```
$ ping 192.168.1.62
```

```
C:\Users\Saso>ping 192.168.1.62
Pinging 192.168.1.62 with 32 bytes of data:
Reply from 192.168.1.62: bytes=32 time<1ms TTL=64
Reply from 192.168.1.62: bytes=32 time<1ms TTL=64
Reply from 192.168.1.62: bytes=32 time<1ms TTL=64
```

Dobiti moramo pozitiven odgovor po zgledu zgornjega izpisa.



Slika 5.5 Pregled ustvarjenih instanc

Nato preverimo še zunanjo SSH-povezljivost instanc. Za to potrebujemo zasebni ključ uporabnika, ki ga generira skripta ostack_create_tenant_user_dns-exercise.sh. Z instanco se povežemo na naslednji način:

```
$ chmod 600 jvzorec-DNS-EX.pem
$ ssh –i jvzorec-DNS-EX.pem root@192.168.1.62
```

S prvim ukazom spremenimo pravice datoteke ključa, tako da je ta dostopna samo lastniku. To je potrebno zaradi varnostnih zahtev klienta SSH. Z drugim ukazom vzpostavimo povezavo SSH z instanco. Če vse deluje pravilno, bi morala biti povezava uspešna. Če

poznamo geslo uporabnika ubuntu, je možno povezavo vzpostaviti tudi brez ključa z naslednjim ukazom:

```
$ ssh ubuntu@192.168.1.62
```

Na ustvarjenih instancah moramo preveriti, ali so pravilno nameščeni programski paketi:

```
ubuntu@jvzorec-dns-ex:~$ dpkg -l | grep bind
ii  bind9                                1:9.8.1.dfsg.P1-4ubuntu0.6   Internet
Domain Name Server
ii  bind9-host                           1:9.8.1.dfsg.P1-4ubuntu0.6   Version of
'host' bundled with BIND 9.X
ii  bind9utils                           1:9.8.1.dfsg.P1-4ubuntu0.6   Utilities
for BIND
ii  libbind9-80                          1:9.8.1.dfsg.P1-4ubuntu0.6   BIND9 Shared
Library used by BIND
ii  python-pycurl                        7.19.0-4ubuntu3              Python
bindings to libcurl
ubuntu@jvzorec-dns-ex:~$

ubuntu@jvzorec-dns-ex:~$ dpkg -l | grep dnsutils
ii  dnsutils                             1:9.8.1.dfsg.P1-4ubuntu0.6   Clients
provided with BIND
ubuntu@jvzorec-dns-ex:~$

ubuntu@jvzorec-dns-ex:~$ ps ax | grep named
 1546 ?        Ssl    0:00 /usr/sbin/named -u bind
 1806 pts/0    S+     0:00 grep --color=auto named
ubuntu@jvzorec-dns-ex:~$
```

Na strežniku DNS morata biti prisotna oba paketa, bind9 in dnsutils, na klientih DNS pa samo dnsutils. Proces `named` mora teči samo na strežniku DNS. V primeru, da stanje na instancah ne ustreza zgoraj opisanemu, se lahko preveri vsebina naslednjih datotek in direktorijev:

- `/var/log/cfn-init.log`
- `/var/log/cloud-init.log`
- `/var/log/cloud-init-output.log`
- `/var/log/heat-provision.log`

- /var/log/part-handler.log
- /var/lib/cloud
- /var/lib/heat-cfntool

6 Sklepne ugotovitve in nadaljnje delo

Izpolnjene so bile glavne zahteve diplomskega dela. S pomočjo komponente Heat in skript v lupini Bash je omogočeno hitro kreiranje virtualne infrastrukture. Infrastrukturo uporabnika je mogoče ustvariti v nekaj minutah in z enim ukazom. Kreirane instance so konfigurirane in povezane z zasebnim in kot parameter podanim zunanjim omrežjem in tako dostopne od zunaj.

Pri delu s komponento Heat pa je kljub temu opazno, da je ta še v hitrem razvoju in se pojavlja še nekaj težav. Tako so bile pri delu opažene težave s kreiranjem skladov in konfiguriranjem varnostnih grup pri instancah. S pomočjo razvijalcev so bili problemi uspešno odpravljeni, končni popravki pa bodo v komponenti Heat odpravljeni v naslednji izdaji projekta OpenStack. Ker je Heat še vedno v razvoju, prav tako ni na voljo uradne dokumentacije, tako da si moramo za zdaj pomagati z informacijami s spletnih forumov razvijalcev in na spletni platformi Launchpad, ki gosti razvijalske informacije, kot so prijavljene napake in načrti funkcionalnosti.

Cilj projekta Heat je popolna združljivost s funkcionalnostjo AWS CloudFormation podjetja Amazon. Trenutno pa še niso podprti vsi viri platforme CloudFormation, zaradi odsotnosti dokumentacije je težko izvedeti, kaj je podprto in kaj ni. Tudi zaradi te funkcionalnosti trenutno Amazon uživa veliko prednost pri ponudbi storitev IaaS v oblaku. Lahko sklepamo in pričakujemo, da bodo preostali ponudniki storitev v oblaku, ki uporabljajo platformo OpenStack in jo ponujajo strankam, zainteresirani za hiter razvoj in čim večjo ponudbo funkcionalnosti. Na komponenti Heat tako dela največ razvijalcev v podjetju Red Hat, ki OpenStack ponujajo svojim strankam.

Obstaja tudi nekaj idej, kako bi se trenutna rešitev lahko še izboljšala. Trenutna rešitev z uporabo skript lupine Bash uporabnikom ni najbolj prijazna, tako da bo verjetno samo kreiranje virtualne infrastrukture moral opravljati administrator v imenu uporabnika, kar se ne sklada popolnoma s paradigmo samopostrežbe v storitvah v oblaku. Alternativno je mogoče omogočiti dostop do skript uporabnikom, kjer pa se pojavi problem izolacije in varnosti. Skripte pri kreiranju ustvarijo zasebni ključ za dostop do instanc. Če so vsi ključi shranjeni v enem direktoriju pod enim uporabniškim imenom, potem en uporabnik lahko dostopa do ključa drugega uporabnika, kar predstavlja precejšnjo varnostno luknjo. Možno bi bilo ločiti posamezne uporabnike po uporabniških imenih na samem sistemu, to pa bi spet pomenilo

ročno kreiranje velikega števila uporabnikov. Najboljša rešitev bi verjetno bil spletni vmesnik, kjer bi posamezni uporabnik lahko kreiral njegovo infrastrukturo z vpisom svojega imena, zasebni ključ za dostop do instanc pa bi nato prejel prek istega spletnega vmesnika, podobno kot je to narejeno v spletnem vmesniku Dashboard. Spletni vmesnik bi nato v ozadju zagnal skripto za kreiranje.

Druga ideja za izboljšavo je povezana s samo implementacijo šablone Heat. Šablona trenutno uporablja vire, ki niso združljivi s platformo AWS CloudFormation. Tu gre predvsem za vire omrežne komponente Quantum. V prihodnosti bi se lahko pojavila tudi potreba, da bi se v primeru prezasedenosti virtualizacijskega sistema v laboratoriju lahko za izvedbo vaj uporabila Amazonova AWS infrastruktura. Ker za zdaj šablona s tem okoljem ni združljiva, to trenutno ni mogoče. Bo pa to verjetno mogoče v prihodnosti, ko bo kompatibilnost med OpenStack Heat in AWS CloudFormation večja in bo na voljo tudi bolj obsežna dokumentacija.

7 Priloge

7.1 Šablona Heat za kreiranje sklada

```
{
  "AWSTemplateFormatVersion" : "2013-05-04",

  "Description" : "AWS CloudFormation template for use with OpenStack. It uses Quantum for networking configuration. It sets up three Ubuntu 12.04 instances, each on its own subnet/network. One instance is set up with bind9 for configuration of DNS server. When stack is created, DNS is not configured, just installed on the instance. The other two instances are used for testing DNS configuration. The three subnets are connected with a router, that is also connected to the external network. Floating IPs are also provided to the spawned instances for external communication.",

  "Parameters" : {
    "KeyName" : {
      "Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instances",
      "Type" : "String"
    },
```

```
      "ExtNetUuid" : {
        "Description" : "UUID of the external network to be used for external access",
        "Type" : "String"
      },

      "InstanceType" : {
        "Description" : "DNSServer EC2 instance type",
        "Type" : "String",
        "Default" : "m1.micro",
        "AllowedValues" : [ "t1.micro", "m1.small", "m1.large", "m1.xlarge", "m2.xlarge",
"m2.2xlarge", "m2.4xlarge", "c1.medium", "c1.xlarge", "cc1.4xlarge", "m1.micro" ],
        "ConstraintDescription" : "must be a valid EC2 instance type."
      },
      "LinuxDistribution": {
        "Default": "U12",
        "Description" : "Distribution of choice",
        "Type": "String",
        "AllowedValues" : "U12"
      }
    },

    "Mappings" : {
      "AWSInstanceType2Arch" : {
        "t1.micro"    : { "Arch" : "32" },
        "m1.small"    : { "Arch" : "32" },
        "m1.large"    : { "Arch" : "64" },
        "m1.xlarge"   : { "Arch" : "64" },
        "m2.xlarge"   : { "Arch" : "64" },
        "m2.2xlarge"  : { "Arch" : "64" },
        "m2.4xlarge"  : { "Arch" : "64" },
        "c1.medium"   : { "Arch" : "32" },
        "c1.xlarge"   : { "Arch" : "64" },
        "cc1.4xlarge" : { "Arch" : "64" },
        "m1.micro"    : { "Arch" : "64" }
      },
      "DistroArch2AMI": {
        "U12"      : { "32" : "U12-i386-cfntools", "64" : "U12-x86_64-cfntools" }
      }
    },

    "Resources" : {
```

```
"network": {
  "Type": "OS::Quantum::Net",
  "Properties": {
    "name": "local_network"
  }
},

"subnet": {
  "Type": "OS::Quantum::Subnet",
  "Properties": {
    "network_id": { "Ref" : "network" },
    "ip_version": 4,
    "cidr": "10.0.10.0/24",
    "allocation_pools": [{"start": "10.0.10.20", "end": "10.0.10.50"}]
  }
},

"DNSServerPort": {
  "Type": "OS::Quantum::Port",
  "Properties": {
    "network_id": { "Ref" : "network" },
    "fixed_ips": [{
      "subnet_id": { "Ref" : "subnet" },
      "ip_address": "10.0.10.30"
    }]
  }
},

"Client1Port": {
  "Type": "OS::Quantum::Port",
  "Properties": {
    "network_id": { "Ref" : "network" },
    "fixed_ips": [{
      "subnet_id": { "Ref" : "subnet" },
      "ip_address": "10.0.10.31"
    }]
  }
},

"Client2Port": {
  "Type": "OS::Quantum::Port",
  "Properties": {
    "network_id": { "Ref" : "network" },
```

```
          "fixed_ips": [{
            "subnet_id": { "Ref" : "subnet" },
            "ip_address": "10.0.10.32"
          }]
        }
      },

      "router": {
        "Type": "OS::Quantum::Router"
      },

      "router_interface_private": {
        "Type": "OS::Quantum::RouterInterface",
        "Properties": {
          "router_id": { "Ref" : "router" },
          "subnet_id": { "Ref" : "subnet" }
        }
      },

      "router_gateway_external": {
        "Type": "OS::Quantum::RouterGateway",
        "DependsOn": "router_interface_private",
        "Properties": {
          "router_id": { "Ref" : "router" },
          "network_id": { "Ref" : "ExtNetUuid" }
        }
      },

      "DNSServerSecurityGroup" : {
        "Type" : "AWS::EC2::SecurityGroup",
        "Properties" : {
          "GroupDescription" : "Enable ping, SSH and DNS (port 53) access",
          "SecurityGroupIngress" : [
            {"IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1", "CidrIp" :
"0.0.0.0/0"},
            {"IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" :
"0.0.0.0/0"},
            {"IpProtocol" : "tcp", "FromPort" : "53", "ToPort" : "53", "CidrIp" :
"0.0.0.0/0"},
            {"IpProtocol" : "udp", "FromPort" : "53", "ToPort" : "53", "CidrIp" :
"0.0.0.0/0"}
          ]
        }
```

```
      },

      "MinimalSecurityGroup" : {
        "Type" : "AWS::EC2::SecurityGroup",
        "Properties" : {
          "GroupDescription" : "Enable only ping and SSH access",
          "SecurityGroupIngress" : [
            {"IpProtocol" : "icmp", "FromPort" : "-1", "ToPort" : "-1", "CidrIp" :
"0.0.0.0/0"},
            {"IpProtocol" : "tcp", "FromPort" : "22", "ToPort" : "22", "CidrIp" :
"0.0.0.0/0"}
          ]
        }
      },

      "DNSServer": {
        "Type": "AWS::EC2::Instance",
        "Metadata" : {
          "AWS::CloudFormation::Init" : {
            "config" : {
              "packages" : {
                "apt" : {
                  "bind9"    : [],
                  "dnsutils" : []
                }
              }
            }
          }
        },
        "Properties": {
          "ImageId"   :   {   "Fn::FindInMap"   :   [   "DistroArch2AMI",   {   "Ref"   :
"LinuxDistribution" },
                           {  "Fn::FindInMap"  :  [  "AWSInstanceType2Arch",  {  "Ref"  :
"InstanceType" }, "Arch" ] } ] },
          "InstanceType"   : { "Ref" : "InstanceType" },
          "KeyName"        : { "Ref" : "KeyName" },
          "SecurityGroups" : [ { "Ref" : "DNSServerSecurityGroup" } ],
          "NetworkInterfaces" : [ { "Ref" : "DNSServerPort" } ],
          "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
            "#!/bin/bash -v\n",
            "echo \"nameserver 8.8.8.8\" >> /etc/resolv.conf\n",
            "/opt/aws/bin/cfn-init\n"
          ]]}}
```

```
        }
      },

      "Client1": {
        "Type": "AWS::EC2::Instance",
        "Metadata" : {
          "AWS::CloudFormation::Init" : {
            "config" : {
              "packages" : {
                "apt" : {
                  "dnsutils" : []
                }
              }
            }
          }
        },
        "Properties": {
          "ImageId"   :   {   "Fn::FindInMap"   :   [   "DistroArch2AMI",   {   "Ref"   :
"LinuxDistribution" },
                           {  "Fn::FindInMap"  :  [  "AWSInstanceType2Arch",  {  "Ref"  :
"InstanceType" }, "Arch" ] } ] },
          "InstanceType"   : { "Ref" : "InstanceType" },
          "KeyName"        : { "Ref" : "KeyName" },
          "SecurityGroups" : [ { "Ref" : "MinimalSecurityGroup" } ],
          "NetworkInterfaces" : [ { "Ref" : "Client1Port" } ],
          "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
            "#!/bin/bash -v\n",
            "echo \"nameserver 8.8.8.8\" >> /etc/resolv.conf\n",
            "/opt/aws/bin/cfn-init\n"
          ]]}}
        }
      },

      "Client2": {
        "Type": "AWS::EC2::Instance",
        "Metadata" : {
          "AWS::CloudFormation::Init" : {
            "config" : {
              "packages" : {
                "apt" : {
                  "dnsutils" : []
                }
              }
```

```
            }
          }
        },
        "Properties": {
          "ImageId"   :   {   "Fn::FindInMap"   :   [   "DistroArch2AMI",   {   "Ref"   :
"LinuxDistribution" },
                            {  "Fn::FindInMap"  :  [  "AWSInstanceType2Arch",  {  "Ref"  :
"InstanceType" }, "Arch" ] } ] },
          "InstanceType"   : { "Ref" : "InstanceType" },
          "KeyName"        : { "Ref" : "KeyName" },
          "SecurityGroups" : [ { "Ref" : "MinimalSecurityGroup" } ],
          "NetworkInterfaces" : [ { "Ref" : "Client2Port" } ],
          "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
            "#!/bin/bash -v\n",
            "echo \"nameserver 8.8.8.8\" >> /etc/resolv.conf\n",
            "/opt/aws/bin/cfn-init\n"
          ]]}}
        }
      },

      "floating_ip_DNSserver": {
        "Type": "OS::Quantum::FloatingIP",
        "DependsOn": "router_gateway_external",
        "Properties": {
          "floating_network_id": { "Ref" : "ExtNetUuid" }
        }
      },

      "floating_ip_Client1": {
        "Type": "OS::Quantum::FloatingIP",
        "DependsOn": "router_gateway_external",
        "Properties": {
          "floating_network_id": { "Ref" : "ExtNetUuid" }
        }
      },

      "floating_ip_Client2": {
        "Type": "OS::Quantum::FloatingIP",
        "DependsOn": "router_gateway_external",
        "Properties": {
          "floating_network_id": { "Ref" : "ExtNetUuid" }
        }
      },
```

```
    "floating_ip_assoc_DNSserver": {
      "Type": "OS::Quantum::FloatingIPAssociation",
      "Properties": {
        "floatingip_id": { "Ref" : "floating_ip_DNSserver" },
        "port_id": { "Ref" : "DNSServerPort" }
      }
    },

    "floating_ip_assoc_Client1": {
      "Type": "OS::Quantum::FloatingIPAssociation",
      "Properties": {
        "floatingip_id": { "Ref" : "floating_ip_Client1" },
        "port_id": { "Ref" : "Client1Port" }
      }
    },

    "floating_ip_assoc_Client2": {
      "Type": "OS::Quantum::FloatingIPAssociation",
      "Properties": {
        "floatingip_id": { "Ref" : "floating_ip_Client2" },
        "port_id": { "Ref" : "Client2Port" }
      }
    }
  },

  "Outputs" : {
  }
}
```

7.2 Skripta ostack_create_tenant_user_dns-exercise.sh

```
#!/bin/bash

KEYSTONE_CMD="/usr/local/bin/keystone"
NOVA_CMD="/usr/local/bin/nova"
QUANTUM_CMD="/usr/local/bin/quantum"
HEAT_CMD="/usr/local/bin/heat"
DNS_TEMPLATE="DNS_exercise.template"
```

```
function usage() {
    echo
    echo "USAGE:$0 tenant/username ext_net_uuid"
    echo "PARAMETERS:"
    echo "   tenant/username - Name to be used for both tenant and username"
    echo "                   - '-DNS-EX' will be appended at the end"
    echo "   ext_net_uuid    - UUID of an existing openstack quantum network"
    echo "                     that will be used for instance external network access"
    echo
}


function get_id () {
    echo `"$@" | awk '/ id / { print $4 }'`
}

if [[ $1 = "--help" || $1 = "-h" || $1 = "help" ]]; then
    usage
    exit 0
fi

if [[ $# -ne 2 ]]; then
    echo "ERROR: Wrong number of parameters"
    usage
    exit 1
fi

# Check if the supplied external quantum network exists, bail out otherwise
${QUANTUM_CMD} net-show $2 2>&1 > /dev/null
if [[ $? -ne 0 ]]; then
    echo "ERROR: Quantum network with the supplied UUID $2 does not exist"
    usage
    exit 1
fi

USERNAME="${1}-DNS-EX"
PASSWD="${USERNAME}"

TENANT_ID=$(get_id ${KEYSTONE_CMD} tenant-create --name ${USERNAME} --enable true ) \
2>&1 > /dev/null
if [[ $? -ne 0 ]]; then
    echo "ERROR: Could not create tenant ${USERNAME}"
    exit 1
```

```
   fi

   ${KEYSTONE_CMD} user-create --name ${USERNAME} --tenant-id ${TENANT_ID} \
   --pass ${PASSWD} --enabled true 2>&1 > /dev/null
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not create user ${USERNAME}"
       exit 1
   fi

   ${NOVA_CMD}   --os-username   ${USERNAME}   --os-password   ${PASSWD}   --os-tenant-name
${USERNAME} \
   keypair-add ${USERNAME}-key > ${USERNAME}-key.pem
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not add keypair for ${USERNAME}"
       exit 1
   fi

   echo "Created user and tenant: ${USERNAME}"
   echo "Password: $PASSWD"
   echo "Private key stored into: ${USERNAME}-key.pem"

   # Temporary workaround for security groups attach bug
   # Just add necessary rules to default security group
   ${NOVA_CMD}   --os-username   ${USERNAME}   --os-password   ${PASSWD}   --os-tenant-name
${USERNAME} \
   secgroup-add-rule default icmp -1 -1 0.0.0.0/0 2>&1 >/dev/null
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not add security group rule for ${USERNAME}"
       exit 1
   fi
   ${NOVA_CMD}   --os-username   ${USERNAME}   --os-password   ${PASSWD}   --os-tenant-name
${USERNAME} \
   secgroup-add-rule default tcp 22 22 0.0.0.0/0 2>&1 >/dev/null
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not add security group rule for ${USERNAME}"
       exit 1
   fi
   ${NOVA_CMD}   --os-username   ${USERNAME}   --os-password   ${PASSWD}   --os-tenant-name
${USERNAME} \
   secgroup-add-rule default tcp 53 53 0.0.0.0/0 2>&1 >/dev/null
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not add security group rule for ${USERNAME}"
       exit 1
```

```
   fi
   ${NOVA_CMD}   --os-username   ${USERNAME}   --os-password   ${PASSWD}   --os-tenant-name
${USERNAME} \
   secgroup-add-rule default udp 53 53 0.0.0.0/0 2>&1 >/dev/null
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not add security group rule for ${USERNAME}"
       exit 1
   fi
   echo "Added security group rules"

   # Use heat to create DNS exercise stack for tenant/user
   ${HEAT_CMD}   --os-username   ${USERNAME}   --os-tenant-name   ${USERNAME}   --os-password
${PASSWD} \
   --os-tenant-id ${TENANT_ID} stack-create ${USERNAME}  -f ${DNS_TEMPLATE} \
   -P "KeyName=${USERNAME}-key;ExtNetUuid=$2"
   if [[ $? -ne 0 ]]; then
       echo "ERROR: Could not create DNS exercise stack for tenant/user ${USERNAME}"
       exit 1
   fi

   echo "Created DNS stack: ${USERNAME}"
   echo
```

7.3 Skripta ostack_stack_list_dns-exercise.sh

```
   #!/bin/bash

   KEYSTONE_CMD="/usr/local/bin/keystone"
   HEAT_CMD="/usr/local/bin/heat"

   TENANTS=$(${KEYSTONE_CMD} tenant-list | grep DNS-EX | awk '{ print $4}')

   for tenant in ${TENANTS}; do
      echo "TENANT: "$tenant
      tenant_id=$(${KEYSTONE_CMD} tenant-list | grep ${tenant} | awk '{ print $2}')
      ${HEAT_CMD}   --os-username   ${tenant}   --os-tenant-name   ${tenant}   --os-password
${tenant} --os-tenant-id ${tenant_id} stack-list
   done
```

7.4 Skripta ostack_purge_tenant_user_dns-exercise.sh

```
#!/bin/bash

KEYSTONE_CMD="/usr/local/bin/keystone"
HEAT_CMD="/usr/local/bin/heat"

function usage() {
    echo
    echo "USAGE:$0 tenant/username"
    echo "PARAMETERS:"
    echo "   tenant/username - Name to be used for both tenant and username"
    echo
}


function get_id () {
    echo `"$@" | awk '/ id / { print $4 }'`
}

if [[ $1 = "--help" || $1 = "-h" || $1 = "help" ]]; then
    usage
    exit 0
fi

if [[ $# -ne 1 ]]; then
    echo "ERROR: Wrong number of parameters"
    usage
    exit 1
fi

USERNAME=${1}
PASSWD=${USERNAME}
TENANT_ID=$(${KEYSTONE_CMD} tenant-list | grep ${USERNAME} | awk '{ print $2 }')

# Delete user's/tenant's stack
${HEAT_CMD}  --os-username  ${USERNAME}  --os-tenant-name  ${USERNAME}  --os-password
${PASSWD} \
    --os-tenant-id ${TENANT_ID} stack-delete ${USERNAME}
if [[ $? -ne 0 ]]; then
    echo "ERROR: Could not delete stack ${USERNAME}"
    exit 1
```

```
fi

# Wait while the stack is being deleted
while :
do
    STACK_STATUS=$(${HEAT_CMD} --os-username ${USERNAME} --os-tenant-name ${USERNAME} \
       --os-password ${PASSWD} --os-tenant-id ${TENANT_ID} \
       stack-show ${USERNAME} 2>&1 2>&1 | grep ' stack_status ' | awk '{ print $4 }')
    echo "stack status: ${STACK_STATUS}"
    if [[ "${STACK_STATUS}" = "DELETE_IN_PROGRESS" ]]; then
        echo "Stack is still being deleted, sleeping for 10s"
        sleep 10
    else
        break;
    fi
done

${KEYSTONE_CMD} tenant-delete ${USERNAME} 2>&1 >/dev/null
if [[ $? -ne 0 ]]; then
    echo "ERROR: Could not purge tenant ${USERNAME}"
    exit 1
fi

${KEYSTONE_CMD} user-delete ${USERNAME} 2>&1 >/dev/null
if [[ $? -ne 0 ]]; then
    echo "ERROR: Could not purge user ${USERNAME}"
    exit 1
fi

echo "Purged user and tenant: ${USERNAME}"
echo
```

7.5 Skripta ostack_purge_all_tenants_dns-exercise.sh

```
#!/bin/bash

KEYSTONE_CMD="/usr/local/bin/keystone"
HEAT_CMD="/usr/local/bin/heat"

TENANTS=$(${KEYSTONE_CMD} tenant-list | grep DNS-EX | awk '{ print $4}')

echo "Purging the following tenants in the background"
```

```
for tenant in ${TENANTS}; do
   echo $tenant
   tenant_id=$(${KEYSTONE_CMD} tenant-list | grep ${tenant} | awk '{ print $2}')
   ./ostack_purge_tenant_user_dns-exercise.sh ${tenant} 2>&1 >/dev/null &
done
```

7.6 Devstack konfiguracijska datoteka localrc

```
ADMIN_PASSWORD=password
MYSQL_PASSWORD=password
RABBIT_PASSWORD=password
SERVICE_PASSWORD=password
SERVICE_TOKEN=tokentoken

# enable quantum

disable_service n-net
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service quantum

# enable heat

ENABLED_SERVICES+=,heat,h-api,h-api-cfn,h-api-cw,h-eng
IMAGE_URLS+=",http://fedorapeople.org/groups/heat/prebuilt-jeos-images/F16-x86_64-
cfntools.qcow2"
```

Seznam slik

Literatura

[1] Peter Mell, Timothy Glance: NIST, National Institute of Standards and Technology, *The NIST Definition of Cloud Computing*, 2011.

[2] M. Hamdaqa, L. Tahvildari, *Cloud computing uncovered: A Research Landscape, Advances in computers vol. 86,* Elsevier, 2012.

[3] *OpenStack Compute Administration Manual, Grizzly 2013.1*, 2013. Dostopno na: http://docs.openstack.org/grizzly/openstack-compute/admin/bk-compute-adminguide-grizzly.pdf

[4] *OpenStack Block Storage Service Administration Guide, Grizzly 2013.1,* 2013. Dostopno na: http://docs.openstack.org/grizzly/openstack-block-storage/admin/bk-block-storage-adminguide-grizzly.pdf

[5] *OpenStack Networking Administration Guide, Grizzly 2013.1*, 2013. Dostopno na: http://docs.openstack.org/grizzly/openstack-network/admin/bk-quantum-admin-guide-grizzly.pdf

[6] *Introducing JSON*. Dostopno na http://www.json.org

[7] *The application/json Media Type for Javascript Object Notation (JSON),* RFC 4627. Dostopno na: http://tools.ietf.org/html/rfc4627

[8] *AWS CloudFormation, User Guide, API Version 2010-05-15.* Dostopno na: http://awsdocs.s3.amazonaws.com/AWSCloudFormation/latest/cfn-ug.pdf