



Sveučilište u Rijeci
TEHNIČKI FAKULTET

OpenStack vježbe za kolegiji: Upravljanje u programskom inženjerstvu

Nositelj: dr. sc. Tihana Galinac Grbac

Autor: Nikola Domazet

Ožujak 2016. Rijeka

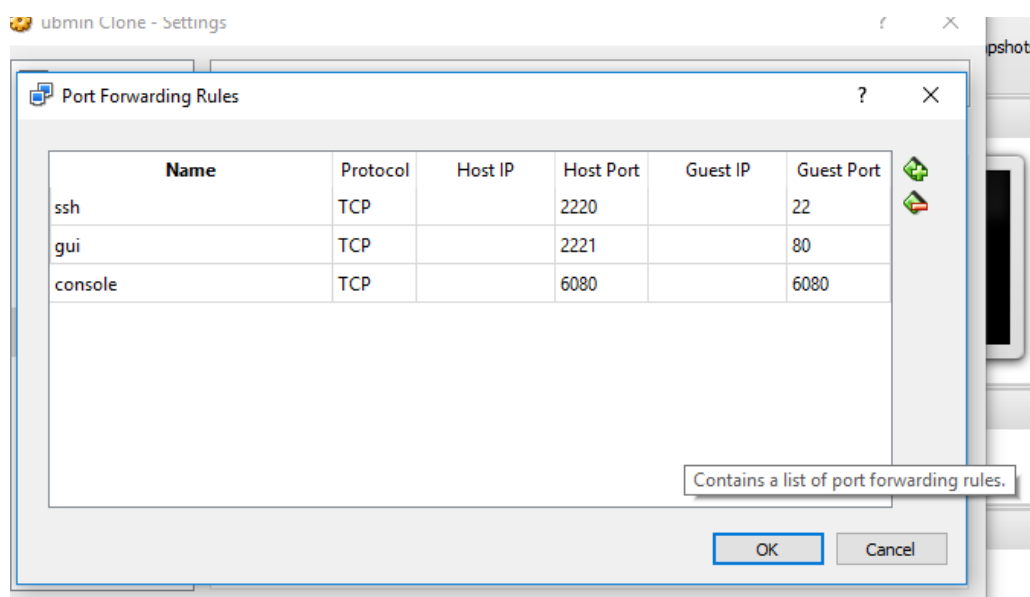
1. Instalacija testnog okruženja

Poglavlje 1. opisuje instalaciju DevStack okruženja na vlastito računalo. Većina vježbi moći će se izvesti pomoću DevStack okruženja. Vježbe koje se moraju izvesti na klasičnoj OpenStack distribuciji biti će naglašene.

VirtualBox

Kao vm hypervisor koristi ćemo VirtualBox. Program se skida sa stranice <https://www.virtualbox.org/wiki/Downloads>. Potrebno je odabrati i skinuti odgovarajući instaler. U slučaju 64 bitnog windowsa to je **VirtualBox 5.0.14 for Windows hosts** (najnovija verzija u trenutku pisanja ovih vježbi). Potrebno je instalirati i pokrenuti Virtualbox. Sliku OS-a virtualnog stroja skinuti ćemo sa stranice <https://help.ubuntu.com/community/Installation/MinimalCD> gdje ćemo odabrati **Ubuntu 15.10 ISO** sliku pod sekcijom **64-bit PC (amd64, x86_64) (Recommended)**.

Pokrenite VirtualBox. Pritisnite gumb **New** u gornjem desnom kutu i pokrenite virtualni stroj. Nazovite ga po želji i postavite type na **Linux** i version na **Ubuntu(64bit)**. Virtualnom stroju dodijelite resurse po želji. Predlažem minimalno 50Gb veličinu diska i oko 4GB ram-a. Ukoliko računalo ima ispod 8gb ram-a dodijelite joj pola ram kapaciteta vašeg računala. Ostale postavke potvrditi bez izmjene. Nakon što smo kreirali virtualni stroj potrebno ga je pritisnuti desnim klikom i otići na **Settings**. U **Settings** pod **System** sučeljem je potrebno otići na **Processor** tab i stroju dodijeliti dodatne procesorske jezgre. Maksimalno je dodijeliti pola kapaciteta ukupnih jezgri računala dok je predloženo dodijeliti 4. Pod **Networking** sučeljem je pod tabom **Adapter1** potrebno postaviti **Attached to NAT**, te je pod **Advanced options** potrebno pritisnuti **Port Forwarding** gumb i postaviti tablicu kao na slici.



Pokrenite virtualni stroj . Biti ćete upitani za start-up disk. Odaberite mini.iso koji ste skinuli u jednom od prethodnih koraka. Pratite instalacijske upute. Kad budete upitani kako particionirati disk, odaberite opciju **Use entire disk**. U jednom koraku instalacije biti ćete upitani koje elemente OS-a instalirati. Sa space tipkom odaberite **Base Ubuntu server** i **OpenSSH** server te sa enter tipkom nastavite instalacijski proces.

Nakon instalacije ponovo će vam biti pružen početni OS instalacijski meni. Potrebno je u prozoru virtualnog stroja u alatnoj traci kliknuti na padajući izbornik **Devices** i pod **Optical Devices** pritisnuti mini.iso kako bismo ISO izbacili iz virtualnog CDROM-a. Nakon toga, potrebno je ponovno pokrenuti virtualni stroj. Ovoga puta bi se trebao pokazati ubuntu login screen.

Kako bi olakšali rad sa virtualnim strojem možemo koristiti komandnu traku glavnog OS sustava na kojemu radimo te putem naredbe:

```
ssh -p 2220 username@localhost
```

(gdje je username korisničko ime koje smo odabrali tijekom instalacije) pristupiti našem virtualnom stroju. Ukoliko koristite linux, naredbu možete pokrenuti unutar terminala. Ako koristite Windows OS predlažem pomoćni alat. Tijekom ove demonstracije biti će korištena MobaXterm aplikacija koja se može skinuti sa <http://mobaxterm.mobatek.net/download-home-edition.html> stranice. Pokrenite MobaXterm aplikaciju i pritisnite **Start local terminal** gumb u centru prozora.

Instalacija DevStack okruženja

DevStack okruženje biti će instalirano na virtualnom stroju stvorenom u prethodnom poglavlju. Prvo je potrebno Za instalaciju DevStack okruženja koristite sljedeće naredbe:

```
git clone https://git.openstack.org/openstack-dev/devstack  
  
cd devstack/
```

Skinite local.conf datoteku sa github-a naredbom:

```
wget --no-check-certificate  
https://github.com/nikoladom91/ARIKS2016/blob/master/Skripte/Heat/resursi/service.php
```

U local.conf datoteci možete zamijeniti postavljene lozinke koje će biti korištene pri instalaciji OpenStack elemenata. Postavljene lozinke su "pass". Kako bi promijenili lozinke otvorite local.conf datoteku naredbom:

```
nano local.conf
```

Zapisani tekst unutar nano text editora se pohranjuje pritiskom na tipke tipkovnice ctrl+x pa y.

Pokrenuti proces instalacije naredbom:

```
./stack.sh
```

Instalacija traje sve do poruke "stack.sh compleated" nakon čega će se otključati terminal.

Pristup DevStack okruženju

Web GUI-u pristupamo putem web adrese localhost:2221 koristeći web preglednik na našem glavnom OS-u unutar kojega je pokrenut VirtualBox.

Za rad sa komandom linijom DevStack-a potrebno je koristiti terminal Virtualnog stroja na kom je okruženje instalirano. Terminalu pristupamo putem prozora VirtualBox prozora virtualog stroja ili putem SSH protokola sa terminala glavnog OS-a.

Važno je naglasiti da DevStack neće preživjeti reboot virtualnog stroja. Pauziranje virtualnog stroja se vrši tako da se u alatnoj traci virtualnog stroja iz padajuće liste File odabere close te se označi **Save the machine state**. Ovako možete zatvoriti VirtualBox bez gašenja OSa virtualnog stroja.

2. Korištenje grafičkog web sučelja Horizon

Poglavlje 2. opisuje način korištenja Horizon usluge. Putem primjera biti će prikazan način korištenja usluga koje OpenStack pruža.

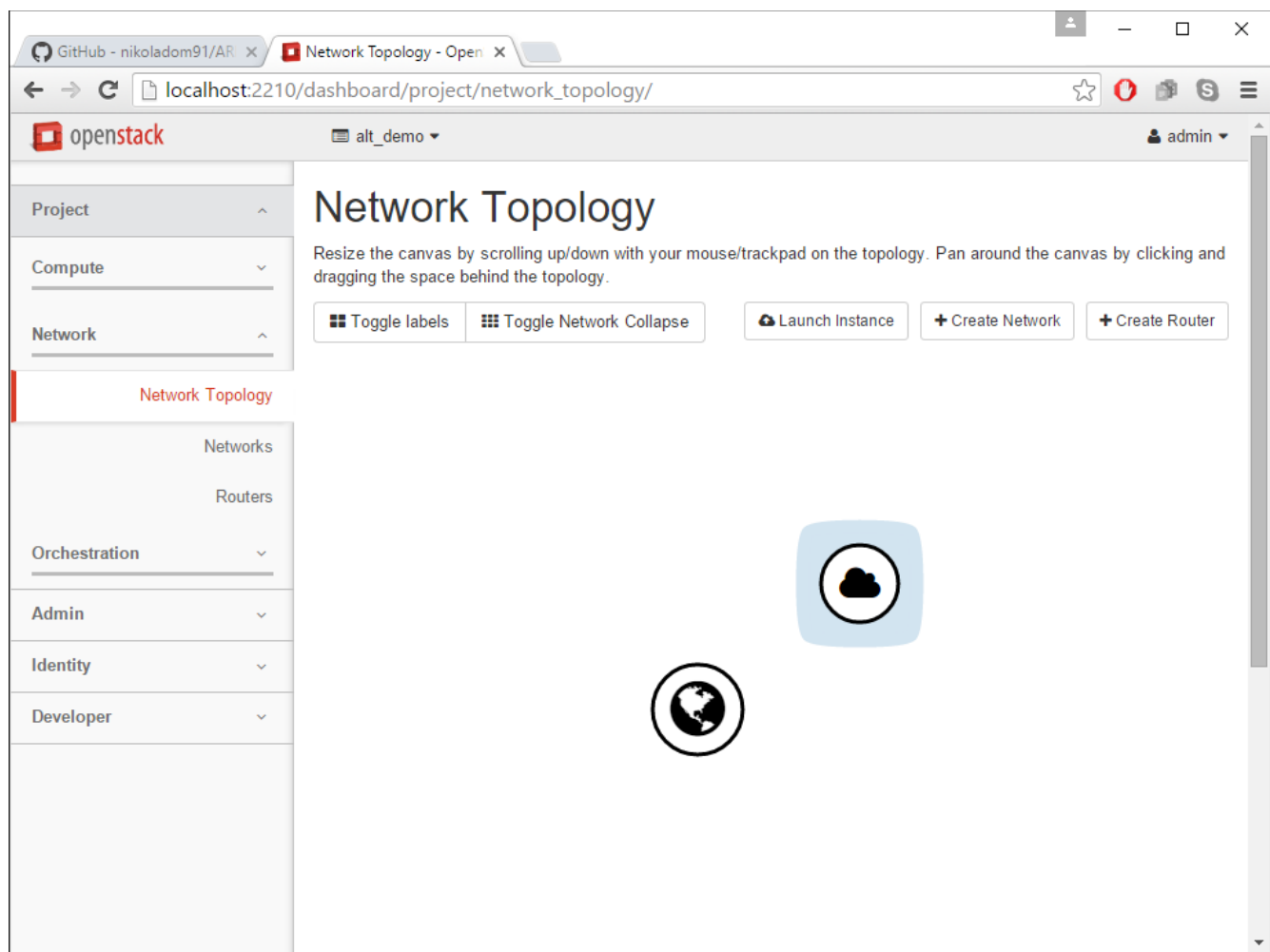
Pristup Horizon sučelju

Horizonu se pristupa putem adrese njegovog web sučelja. U lokalnoj distribuciji OpenStack usluge (npr. unutar SEIP laboratorija), ta adresa je uobičajeno adresa kontrolera. U DevStack izvedbi opisanoj u prethodnim primjerima ta adresa je "localhost:2221". Korisnik putem ponuđene forme upisuje korisničko ime i lozinku te vrši prijavu u sustav. Ovlasti su dodijeljene ovisno o korištenom korisničkom računu.

Stvaranje privatne mreže

Kako bi instance (virtualni strojevi) imale pristup mreži, potrebno je kreirati vlastitu virtualnu lokalnu mrežu. Kreacija i upravljanje mrežom se vrše putem **Network** sučelja. Njemu se pristupa preko istoimenoga elementa iz padajućeg izbornika lociranog na lijevom kraju Horizon sučelja. Network sučelje je dio **Project** izbornika.

OpenStack okruženje kao dio svoje strukture ima definiranu mrežu koja ima pristup vanjskim mrežama odnosno internetu. Kako bi naše instance bile u mogućnosti komunicirati međusobno, potrebno je kreirati lokalnu mrežu. To činimo tako da u sučelju **Network \ Network Topology** pritisnemo gumb **Create Network**. Putem ponuđenih formi kreiramo virtualnu mrežu i njezin subnet. Mrežu ćemo nazvati my_net1 i Admin State ćemo postaviti na UP. Subnet mrežu ćemo nazvat my_sub1, ip adresu ćemo postaviti na 10.20.0.0/24, ip verziju postaviti na Ipv4 a Gateway IP postaviti na 10.20.0.1. DHCP ćemo uključiti te u polje DNS Name Servers upisati vrijednosti "8.8.8.8" i "8.8.4.4", svaku u vlastitome redu. Ispunjenjem ovih formi smo kreirali my_net1 virtualnu mrežu i njezin subnet te na taj subnet sada možemo spojiti naše instance. Iz prikaza topologije na sučelju **Network \ Network Topology** vidimo našu novu mrežu te njezinu povezanost s ostatkom topologije. Iz dolje prikazane slike je vidljivo da naša mreža nije povezana sa drugim mrežama. Instance spojene na ovakvu mrežu moći će jedino međusobno komunicirati preko lokalnih adresa.



Privatnu my_net1 mrežu moramo spojiti sa javnom mrežom putem virtualnog routera kako bi instance mogle pristupati internetu. Njega kreiramo preko gumba **Create Router** u sučelju **Network \ Network Topology**. Ponuđenu formu ispunimo imenom routera, my_router1, te mu Admin State postavimo na UP a kao External Network odaberemo javnu mrežu. Ovime smo stvorili router koji služi kao gateway prema javnoj mreži. Na njegova sučelja sada možemo spajati naše privatne mreže. To činimo tako da u sučelju **Network \ Routers** pritisnemo na ime routera kojemu želimo dodati mrežno sučelje. Nakon toga je potrebno odabrati tab **Interfaces** gdje će nam biti ponuđen gumb **Add Interface**. Pritiskom na njega će nam se otvoriti forma unutar koje ćemo odabrati subnet koji želimo spojiti na to mrežno sučelje, my_sub1, te njegovu IP adresu, istu kao IP adresu gateway-a odabranog subneta odnosno 10.20.0.1. Ovime smo završili podešavanje našeg virtualnog routera.

Izvršavanjem navedenih koraka kreirali smo vlastitu subnetiranu mrežu koja ima pristup javnoj mreži. Instance spojene na ovu privatnu mrežu biti će u mogućnosti komunicirati međusobno i s internetom ukoliko mu javna mreža ima pristup. Pogled na topologiju unutar sučelja **Network \ Network Topology** bi trebao odgovarati idućoj slici:

GitHub - nikoladom91/AR

Network Topology - Open

localhost:2210/dashboard/project/network_topology/

openstack

alt_demo

admin

Project

Compute

Network

Network Topology

Networks

Routers

Orchestration

Admin

Identity

Developer

Network Topology

Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.

Toggle labels

Toggle Network Collapse

Launch Instance

Create Network

Create Router

8

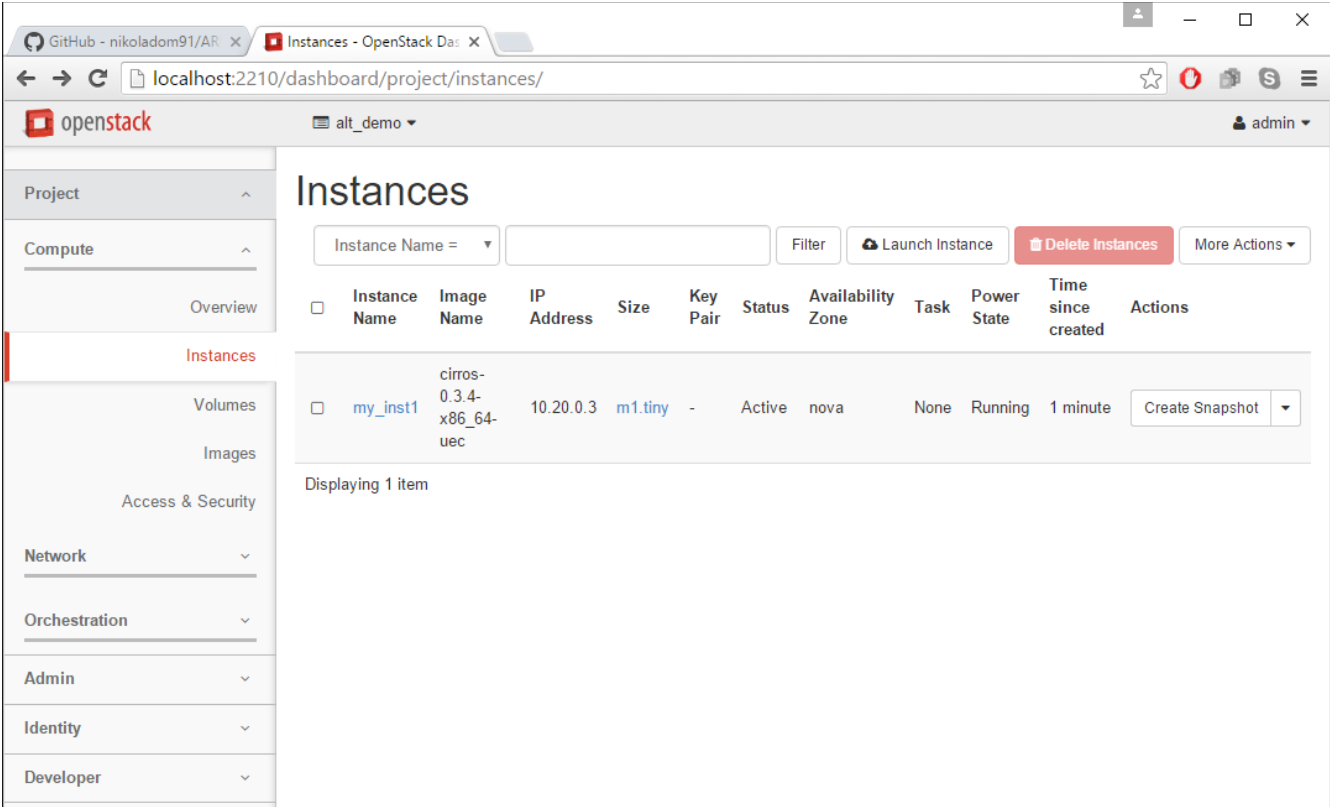
Stvaranje instance

Instance su virtualni strojevi koje OpenStack pogoni. Njima rukujemo putem **Compute** sučelja koje je dio **Projec** izbornika.

Kako bi stvorili novu instancu potrebno je unutar sučelja **Compute / Instances** pritisnuti gumb **Launch Instance** te ispuniti ponuđenu formu. Pod tabom **Details** Availability Zone postavimo u "nova", instancu nazovemo my_inst1 te Instance Count postavimo na 1. Tab **Flavour** postavimo na "m1.tiny". Pod tabom **Source** "Select Boot Sources" odaberemo "Image" te iz ponuđenog izbornika izaberemo "cirros-0.3.4-x86_64-uec" os sliku.

Odabir mreže na koju će instanca biti spojena može se prilikom kreacije definirati pod tabom **Networks**. Unutar ovog taba možemo odabrati na koje će subnet biti spojena naša instanca. Iz sekcije Available dodajemo mreže kojima će naša instanca imati pristup putem automatski kreiranih virtualnih mrežnih sučelja. U našem slučaju odabiremo mrežu my_net1. Tijekom kreacije instance također možemo definirati dodatne sigurnosne postavke. Njih odabiremo na isti način kao i mreže putem tabova **Security Groups** i **Key Pair**. Unutar ovih tabova instanci možemo dodati sigurnosni ključ i definirati koje sigurnosne grupe se odnose na instancu. U našem slučaju Key Pair ostavljamo praznim a Security Groups odabiremo default.

Ovako ispunjena forma će stvoriti instancu koja će biti vidljiva unutar sučelja **Compute / Instances**.



The screenshot shows the OpenStack dashboard interface. The top navigation bar includes the OpenStack logo, a dropdown menu for 'alt_demo', and a user profile for 'admin'. The left sidebar contains a navigation menu with categories like 'Project', 'Compute', 'Instances', 'Volumes', 'Images', 'Access & Security', 'Network', 'Orchestration', 'Admin', 'Identity', and 'Developer'. The 'Compute' section is expanded, showing 'Overview' and 'Instances'. The 'Instances' page title is 'Instances'. Below the title, there are filters for 'Instance Name', a 'Filter' button, and action buttons for 'Launch Instance', 'Delete Instances', and 'More Actions'. A table lists the instances with columns: Instance Name, Image Name, IP Address, Size, Key Pair, Status, Availability Zone, Task, Power State, Time since created, and Actions. One instance, 'my_inst1', is listed with the image 'cirros-0.3.4-x86_64-uec', IP address '10.20.0.3', size 'm1.tiny', key pair '-', status 'Active', availability zone 'nova', task 'None', power state 'Running', and time '1 minute'. The 'Actions' column for this instance has a 'Create Snapshot' button. Below the table, it says 'Displaying 1 item'.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
my_inst1	cirros-0.3.4-x86_64-uec	10.20.0.3	m1.tiny	-	Active	nova	None	Running	1 minute	Create Snapshot

Kako bi instanci mogli pristupati sa vanjske mreže potrebno joj je dodijeliti Floating IP adresu. Floating IP adresa je jedna od rezerviranih IP adresa u prostoru javne mreže. Ovime omogućujemo uređajima koji nisu unutar naše virtualne lokalne mreže da komuniciraju sa instancom. Unutar sučelja **Compute / Instances** u redu instance kojoj želimo dodijeliti Floating IP potrebno je kliknuti na gumb **Associate Floating IP** koji se nalazi u polju **Actions**. Ukoliko gumb ne pokazuje taj tekst potrebno je pritisnuti gumb ▼ pored tog gumba te iz padajućeg izbornika odabrati element **Associate Floating IP**.

U ponuđenoj formi pod poljem IP Adress potrebno je odabrati Floating IP kojeg želimo iskoristiti. U padajućoj listi se nalaze sve Floating IP adrese dodijeljene našem korisničkom računu koje nisu u uporabi. Ukoliko nemamo slobodne adrese potrebno je alocirati novu putem + gumba. Uz to je potrebno iz padajućeg izbornika **Port to be associated** odabrati mrežno sučelje naše instance koje želimo povezati sa Floating IP adresom. Nakon ovog postupka našoj instanci možemo pristupiti preko njezine Floating IP adrese s vanjskih konzola.

Našoj instanci možemo pristupiti na dva načina:

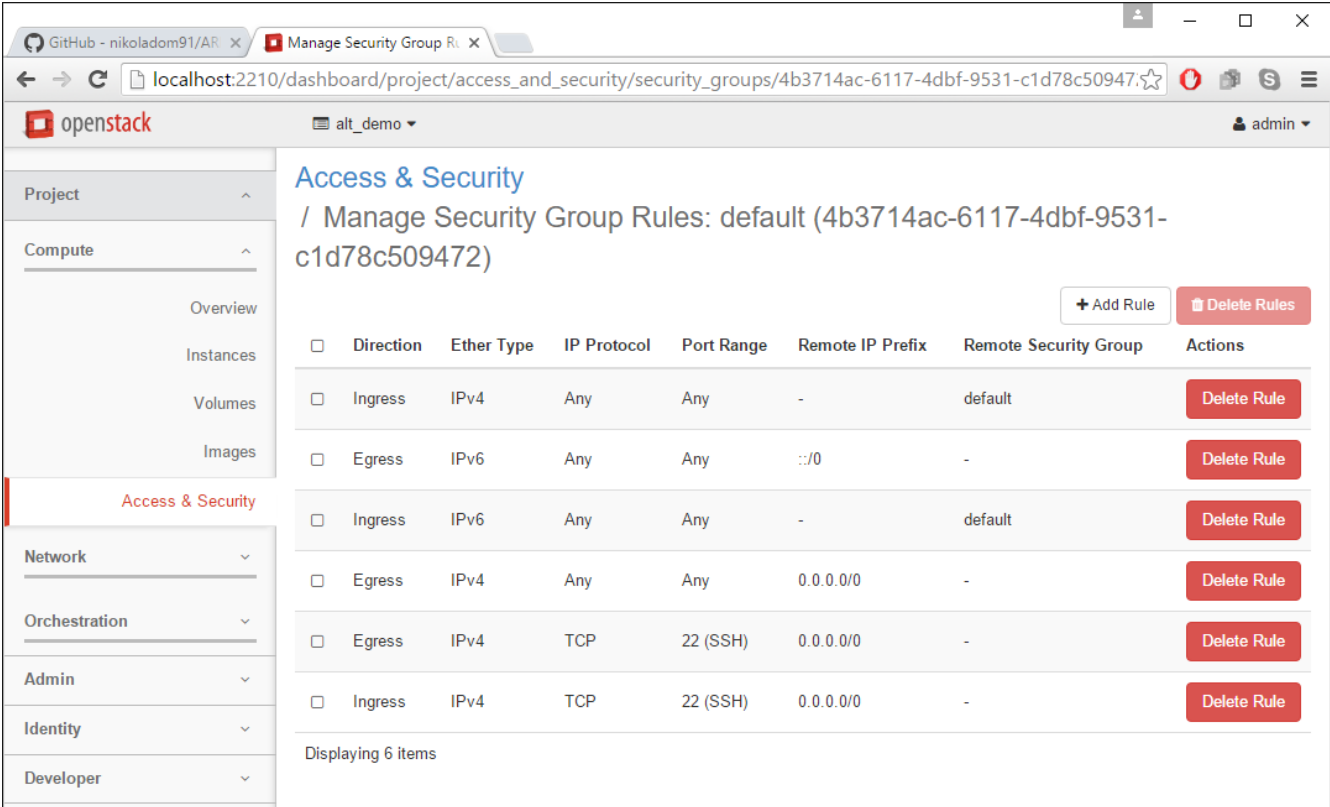
- Najjednostavniji ali i najograničeniji način je putem Horizon GUI-a. Unutar **Compute / Instances** sučelja pritisnemo ime instance kojoj želimo pristupiti te potom odaberemo tab **Console**. Unutar Horizon-a će nam se pokazati simulacija ispisa ekrana naše instance. Pritiskom na taj ekran možemo simulirati korištenje instance putem miša i tipkovnice.
- Drugi način pristupa instanci vrši se putem njezine mrežne veze. Ukoliko smo ispravno postavili sigurnosna pravila, našoj instanci možemo pristupiti preko njezine Floating IP adrese protokolima kao što je SSH. Ukoliko korišteni protokol zahtijeva sigurnosni ključ, možemo iskoristiti .pem datoteku ključa koja je preuzeta tijekom stvaranja ključa koji je dodijeljen instanci. Stvaranje sigurnosnog ključa i podešavanje sigurnosnih pravila opisano je u idućem poglavlju.

Sigurnosne postavke

Sigurnosnim postavkama rukujemo putem **Compute / Access & Security** sučelja.

Upravljanje sigurnosnim grupama:

Kako bi mijenjali postavke sigurnosnih grupa potrebno je pod tabom **Security Groups** u redu grupe koju želimo izmijeniti, default, pritisnuti gumb Manage Rules. Unutar prikazanog sučelja možemo vidjeti trenutna sigurnosna pravila te ih brisati i dodavati nova. Kako bi omogućili SSH komunikaciju s našim instancama, potrebno je grupi dodati dva nova pravila. Pravila se dodaju gumbom **+Add Rule**. U formi je potrebno postaviti Rule kao Custom TCP Rule, smjer kao Ingress te postaviti broj porta kao 22. Ovime Instancama u sigurnosnoj grupi **default** omogućujemo da primaju poruke preko porta 22. Nakon što dodamo dodatno pravilo, ovoga puta Egress smjera te istim ostalim postavkama, omogućili smo SSH komunikaciju sa instancama.



The screenshot shows the OpenStack dashboard interface. The left sidebar contains navigation links for Project, Compute, Network, Orchestration, Admin, Identity, and Developer. The main content area is titled 'Access & Security' and shows the 'Manage Security Group Rules' page for the 'default' security group. The page includes a table of rules and buttons for adding and deleting rules.

<input type="checkbox"/>	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
<input type="checkbox"/>	Ingress	IPv4	Any	Any	-	default	Delete Rule
<input type="checkbox"/>	Egress	IPv6	Any	Any	::/0	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv6	Any	Any	-	default	Delete Rule
<input type="checkbox"/>	Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
<input type="checkbox"/>	Egress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	Delete Rule

Displaying 6 items

Kreacija sigurnosnog ključa:

Kako bi kreirali novi sigurnosni ključ potrebno je pod Key Pairs tabom pritisnuti gumb **+Create Key Pair**. Ključ ćemo nazvati "my_key1". Ključ je kreiran i dodan u listu ključeva te je kopija stvorenoga ključa automatski preuzeta od strane internet preglednika putem kojega pristupamo Horizon-u. Listu kreiranih sigurnosnih ključeva vidimo pod tabom Key Pairs. Preuzetoj datoteci ključa je potrebno promijeniti dopuštenja tako da odemo u direktorij unutar kojega se nalazi te primijenimo naredbu:

```
chmod 600 my_key1.pem
```

Kreacija OS slike putem Glance usluge

Kako bi iskoristili specifičnu OS sliku prilikom kreacije naše instance, tu sliku je potrebno dodijeliti OpenStack sustavu. Slike se dodaju putem **Compute \ Images** sučelja pritiskom na gumb **+Create Image**. Sliku ćemo nazvati "Ubuntu_Cloud" te joj izvor postaviti na Image Location. U Image Location polje ćemo unijeti url "<https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img>" koji pokazuje na web lokaciju ubuntu cloud slike. Ukoliko se sustavu želi dodati slika pohranjena na računalu, potrebno je pod Image Source odabrati Image File. Kao format slike odabiremo "QCOW2 – QEMU Emulator" te završavamo kreaciju slike. Ovu sliku ćemo u idućim koracima moći koristiti prilikom kreacije naše Instance.

Pristup CLI

Kako bi smo mogli pristupiti CLI OpenStack-a potrebno je environment varijable postaviti na pravilne vrijednosti kako bi OpenStack iz njih mogao očitati potrebne informacije. To možemo učiniti ručno no OpenStack nam omogućuje da automatiziramo taj proces. Putem Horizona preko sučelja **Compute \ Access & Security** pod **API Access** tabom pritiskom na gumb **Download OpenStack RC File v2.0** dohvaćamo skriptu čije će pokretanje automatski dodijeliti potrebne vrijednosti environmenta varijablama. Skriptu pokrećemo putem naredbe:

```
source naziv_skripte.sh
```

gdje je naziv_skripte zamijenimo imenom skinute datoteke. Prilikom pokretanja skripte biti ćemo pitani te moramo upisati našu OpenStack lozinku. Ovime vršimo autentifikaciju te nakon ovog koraka možemo koristiti sve CLI naredbe OpenStack-a za koje, putem izvršene autentifikacije, imamo dozvole.

Vježbe

Vježbe u daljnjim tekstu su podijeljene u dvije grupe. Heat vježbe i Python vježbe. Skripte korištene za izvršavanje tih vježbi su cijele ili djelomično prikazane u samom opisu vježbe. Samoj skripti možete pristupiti putem github-a. Github repozitorij na adresi <https://github.com/nikoladom91/ARIKS2016> sadrži mapu "Skripte". Unutar njega se nalaze mape Heat i Python. Svaka mapa sadrži relevantne datotke. Skripta za python vježbu 1 će se znači nalaziti unutar Skripte/Python/Skripta1.py datoteke. Predloženo je da se cjeli repozitorij skine na DevStack virtualni stroj putem naredbe:

```
git clone https://git.openstack.org/nikoladom91/ARIKS2016
```

3. Heat

Kako bi Heat skripte radile, pretpostavljeno je da su obavljene operacije u vježbama opisane pod [Upravljanje sigurnosnim grupama](#), [Kreacija sigurnosnog ključa](#) i [Kreacija OS slike putem Glance usluge](#).

Heat komponenta OpenStack-a se koristi kako bi automatizirali određene procese. Kroz iduće dvije vježbe ćemo prikazati načine na koje pokrećemo Heat skripte i objasniti njihovu sintaksu. Skripte su pisane unutar .yaml datoteka i prate TOSCA format. Za detaljnije informacije o standardu i uvid u detaljnije objašnjenje sintakse proučite stranicu na linku:

<http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd03/TOSCA-Simple-Profile-YAML-v1.0-csd03.html>

Heat skripte, također zvane stacks, služe kako bi se određeni procesi unutar OpenStack-a mogli automatizirati. Pomoću njih definiramo akcije koje će biti prevedene u slijed naredbi koje OpenStack može izvršiti. Skripte možemo pokrenuti na dva načina: putem OpenStack CLI i putem Horizon web sučelja.

Pristup Heat usluzi

Putem CLI:

Pristup OpenStack CLI-u je objašnjen u poglavlju [Pristup CLI](#).

Naredba za pokretanje Heat skripte iz OpenStack CLI je:

```
heat stack-create ime_stack_a -f heat_skripta.yaml
```

Naredba za prikaz stanja stack-a je:

```
heat stack-show ime_stack_a
```

Naredba za prikaz liste svih stack-ova:

```
heat stack-list
```

Putem Horizon sučelja:

Putem Horizon web sučelja potrebno je pristupiti Orchestration sučelju koje je dio "Project" izbornika. U sučelju **Orchestration \ Stacks** nalazi se gumb Launch Stack. Pritiskom na taj gumb stvara se forma. Pod resource je potrebno unijeti lokaciju Heat skripte. Pritiskom gumba Start skripta počinje sa izvršavanjem. Ukoliko skripta očekuje dodatan unos od korisnika, otvori će se nova specifična forma koju je potrebno ispuniti prije pokretanja skripte. Status pojedinih stack-ova može se promatrati putem **Orchestration \ Stacks** sučelja.

Heat vježba 1

Opis skripte 1:

Skripta 1 automatski kreira instancu unutar OpenStack-a. Sve informacije potrebne za kreaciju instance su zapisane izravno u skripti.

Skripta 1:

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

resources:
  my_instance2:
    type: OS::Nova::Server
    properties:
      image: ubuntu_cloud
      flavor: m1.small
      key_name: my_key1
      networks:
        - network: private-net
```

Opis komponenata skripte 1:

<code>heat_template_version</code>	Obavezna sekcija koja definira verziju sintakse korištenu u skripti, Skripta 1 koristi 2013-05-23, prvu puštenu i najčešće korištenu verziju.
<code>description</code>	Opcionalna sekcija koja sadrži opis skripte.
<code>Resources</code>	Objekti koje kreiramo i povezujemo. Unutar TOSCA formata oni se smatraju čvorovima. U ovom primjeru stvaramo jedan objekt odnosno samostalan (ne povezan s ničim) čvor. Svaki objekt je definiran svojim imenom, u ovom slučaju <code>my_instance2</code> .
<code>type</code>	Definira tip resursa. <code>my_instance2</code> je definiran kao <code>OS::Nova::Server</code> . To je vrsta resursa specifična za OpenStack čija je glavna funkcija kreacija instance putem Nova aplikacije
<code>Properties</code>	Svojstva vezana za resurs. Svaka vrsta resursa ima određena svojstva čije vrijednosti moraju biti definirane kako bi resurs mogao biti ispravno izvršen. U slučaju <code>OS::Nova::Server</code> resursa, ta svojstva opisuju vrijednosti potrebne za kreaciju instance kao: korištena slika, korišteni flavor, korišteni sigurnosni ključ i mreža na koju će instance biti spojena

Heat vježba 2

Opis skripte 2:

Unutar skripte 1 podatci potrebni za kreaciju resursa bili su napisani unutar koda. Skripta 2 od korisnika preuzima potrebne podatke prilikom pokretanja. Ovime postizemo da se funkcionalnost skripte može modificirati na predviđen način te eliminiramo potrebu za mijenjanjem koda skripte te time uvođenjem mogućih grešaka. Skripta 2 također pruža izlaznu informaciju koja korisniku javlja određene podatke o kreiranom resursu. U ovome slučaju korisnik je obaviješten o IP adresi pokrenute instance.

Skripta 2:

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

parameters:
  image:
    type: string
    label: Image name or ID
    description: Image to be used for compute instance
    default: cirros-0.3.3-x86_64
  flavor:
    type: string
    label: Flavor
    description: Type of instance (flavor) to be used
    default: m1.small
  key:
    type: string
    label: Key name
    description: Name of key-pair to be used for compute instance
    default: my_key1
  private_network:
    type: string
    label: Private network name or ID
    description: Network to attach instance to.
    default: private-net

resources:
  my_instance3:
    type: OS::Nova::Server
    properties:
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key }
      networks:
        - network: { get_param: private_network }

outputs:
  instance_ip:
    description: IP address of the instance
    value: { get_attr: [my_instance3, first_address] }
```


Opis komponenata skripte 2:

`parameters`

Sekcija koda koja definira parametre koje korisnik mora unijeti. Sačinjena je od pojedinih parametara te njihovih atributa. Svaki parametar je definiran svojim imenom te su njegovi atributi zapisani u sub-sekcijama. Parametar `image` koristi attribute: `type` koji opisuje kakvog je tipa parametar (`string`), `label`, `description` koji dodatno opisuje parametar i `default` koji sadrži standardnu korištenu vrijednost ukoliko atribut nije unesen.

`Outputs`

Definira koju informaciju prikazati nakon što je stack izvršen. Komponente su pojedini izlazi. Skripta 2 ima samo jedan izlaz, `instance_ip`, koji dohvaća ip adresu kreirane instance. Sub-sekcija pojedinog izlaza definira `description` koji opisuje izlaz te `value` koji sadrži dohvaćenu vrijednost izlaza.

`{ get_param: par }`

Sintaksa za korištenje vrijednosti parametra unutar skripte. Umjesto `par` upisuje se ime parametra kojeg dohvaćamo. U skripti 2 tako dohvaćamo nazive OS slike `image`, dodijeljenih resursa `flavor`, sigurnosni ključ `key` i naziv mreže `private_network` koje koristimo pri pokretanju resursa.

`{ get_attr: [res, attr] }`

Sintaksa za dohvaćanje i korištenje atributa resursa. Za razliku od parametara koji predstavljaju vrijednosti koje je korisnik stvorio, atribut predstavlja vrijednosti elemenata koji čine resurs kao što je u našem slučaju IP adresa instance. Pod `res` upisujemo naziv resursa kojemu dohvaćamo atribut te pod `attr` upisujemo naziv atributa koji želimo dohvatiti.

Heat vježba 3

Opis skripte 3:

Unutar skripte 3 stvaramo tri resursa. Dvije instance te jedan random number generator (rng). Resurs `inst_simple` će, uz samo pokretanje instance, na pokrenutoj instanci izvršiti definirani start-up kod. U ovom slučaju to će biti kreacija `hello.txt` datoteke unutar koje ćemo upisati string "Hello, World!". Resurs `inst_advanced` također pokreće predefinirani start-up kod no ovaj puta koristi vrijednosti dohvaćene putem Heat skripte. Prva sekcija ispod prikazanog koda prikazuje resurs `rng` dok druga i treća sekcija prikazuju dio koda `inst_simple` i `inst_advanced` koji opisuje definiranje i pokretanje start-up koda. Ovako definiran start-up kod pokreće se kao root tijekom inicijalizacije instance putem `cloud-init` pristupa koji je dio većine cloud OS slika (npr. Ubuntu cloud image).

Djelovi skripte 3:

```
...
resources:
  rng:
    type: OS::Heat::RandomString
    properties:
      length: 4
      sequence: digits

...
inst_simple:
  type: OS::Nova::Server
  properties:
    ...
    user_data_format: RAW
    user_data: |
      #!/bin/sh
      echo "Hello, World!" >> hello.txt

...
inst_advanced:
  type: OS::Nova::Server
  properties:
    ...
    user_data_format: RAW
    user_data:
      str_replace:
        params:
          __name__: { get_param: name }
          __rnum__: { get_attr: [rng, value] }
        template: |
          #!/bin/sh
          echo "Hello, my name is __name__. Here is a random number:
__rnum__." >> hello.txt
...

```

Opis komponenata skripte 3:

<code>user_data_format</code>	Svojstvo koji definira format unesenog koda. Korištenjem RAW formata kod proslijeđujemo bez dodatnih modifikacija.
<code>user_data</code>	Sekcija unutar koje definiramo start-up kod koji se pokreće prilikom pokretanja instance. Možemo direktno unijeti kod ili definirati skriptu unutar koje se nalazi kod.
<code>str_replace</code>	Ukoliko želimo koristiti promjenjive vrijednosti dobivene putem Heat-a unutar <code>user_data</code> prilažemo ovu sub sekciju. Unutar nje moramo definirati dodatne sub sekcije <code>params</code> i <code>template</code> . Pod <code>params</code> definiramo ključne riječi koje želimo zamijeniti određenim vrijednostima dobivenima putem Heat-a kao što su parametri i atributi. Pod <code>template</code> upisujemo željeni startup-kod. Sve ključne riječi unutar unutar ovog start-up koda će prilikom izvođenja biti zamijenjene vrijednostima definiranim unutar <code>params</code> sekcije.

Heat vježba 4

Heat vježba 4 ne može se vršiti unutar DevStack okruženja. Vježbu je potrebno izvršiti unutar klasične OpenStack distribucije.

Opis skripte 4:

Unutar skripte 4 koristimo vlastite tipove resursa te bolje definiramo veze između resursa. U prethodnim primjerima smo definirali da sekcija `type` definira vrstu resursa koji će biti stvoren. Do sada smo isključivo koristili tipove resursa definirane Heat aplikacijom. Ukoliko kao `type` upišemo lokaciju `.yaml` datoteke kreirati ćemo vlastiti tip resursa. Naš tip resursa biti će Heat skripta koju pozivamo. Radi jednostavnijeg opisa, skriptu koja poziva novi tip zvati ćemo nad-skripta dok ćemo skriptu koja čini novi tip resursa zvati pod-skripta.

Pod-skripta je standardna Heat skripta kao bilo koja od skripti iz prethodnih primjera. Jedina razlika je u tome što pod-skriptu umjesto korisnika poziva nad-skripta. Kako nad-skripta poziva pod-skriptu, ona joj mora biti u mogućnosti proslijediti relevantne vrijednosti koje pod-skripta očekuje kao unos od korisnika unutar svojih parametara. Nad-skripta to vrši putem `properties` sekcije. U skripti 1 je opisano da putem te sekcije Heat definira vrijednosti svojstava vezanih za resursa kao što su OS slika prilikom stvaranja instance u `OS::Nova::Server` tipu resursu. Nad-skripta će u slučaju pokretanja pod-skripte elemente svoje `properties` sekcije proslijediti istoimenim elementima `parameters` sekcije podskripte. Na taj način se vrši prosljeđivanje informacije između različitih razina Heat skripti. Ovime možemo napisati kompleksnije skripte te definirati međusobne odnose između raznih resursa. Ovo između ostaloga u ovoj implementaciji TOSCA modelu predstavlja vezu između čvorova.

Prethodno opisane poveznice povezuju resurse vezane za Heat platformu. Kako bi definirali veze među aplikacijama koje će se vršiti na našim instancama potrebno je koristiti `user_data` opisan u skripti 3. Skripta 4 implementira automatizirano kreiranje dviju instanci. Prva instanca pokreće i konfigurira `mysql` bazu podataka dok druga pokreće `wordpress` web uslugu koja koristi bazu prve instance. Aplikacije su automatski instalirane putem start-up koda definiranog putem `user_data` sekcije te su prilikom konfiguracije `wordpress` usluge korišteni atributi `mysql` resursa koji su sadržavali potrebne informacije kako bi se uspostavila veza. Dohvaćeni atributi su IP adresa instance na kojoj se pokreće `mysql` te podatci korišteni tijekom kreacije korisnika `mysql` baze.

Pokretanjem start-up koda kao definiranog u skripti 3 Heat nema mogućnost nadzora nad njegovim izvršavanjem. Nakon što je start-up kod prosljeđen `cloud-init` mehanizmu, Heat smatra instancu spremnom iako bi izvršenje start-up koda moglo duže potrajati. Ukoliko nam dovršenje start-up koda utječe na druge resurse potrebno je implementirati mehanizam čekanja putem kojega će se resurs smatrati spremnim tek kada start-up skripta bude izvršena.

Mehanizam čekanja implementiramo putem **OS::Heat::WaitCondition** tipa resursa. Ispod prikazani isječci koda mysql.yaml skripta prikazuju njegovu implementaciju. Svojstvo handle sadrži element putem kojega ćemo rukovati ovim resursom, svojstvo count predstavlja broj signala koje očekujemo, a svojstvo timeout predstavlja vrijeme nakon kojega se resurs smatra neuspješno uspostavljenim ukoliko nije primljen definirani broj signala. Kako skriptu mysql.yaml unutar skripte 4 koristimo kao vlastiti resurs, ukoliko wait_condition resurs javi grešku, cjeli mysql.yaml resurs će biti smatran neuspješno uspostavljenim.

Resursom wait_condition resurs rukujemo putem **OS::Heat::WaitConditionHandle** tipa resursa. Ovaj resurs definira curl poziv koji možemo izvršiti unutar naše start-up skripte. U ispod prikazanom kodu, navedeni curl poziv je stavljen na dno start-up koda te će ga cloud-init izvršiti nakon što završi sa prethodnim naredbama. Izvedbom te naredbe, curl poziv će wait_condition resursu poslati signal JSON formata (u ovome slučaju "status": "SUCCESS") koji će inkrementirati brojač primljenih signala. Kada brojač dostigne predviđenu vrijednost wait_condition se smatra izvršenim te se cjelokupni resurs, u ovom slučaju mysql.yaml, može smatrati ispravno uspostavljenim.

Dio mysql.yaml skripte koju skripta 4 poziva:

```
...
resources:
  ...

  wait_condition:
    type: OS::Heat::WaitCondition
    properties:
      handle: { get_resource: wait_handle }
      count: 1
      timeout: 600

  wait_handle:
    type: OS::Heat::WaitConditionHandle

  ...
  mysql_instance:
    type: OS::Nova::Server
    properties:
      ...
      user_data:
        str_replace:
          params:
            ...
            wc_notify: { get_attr: ['wait_handle', 'curl_cli'] }
      template: |
        #!/bin/bash -ex
        ...
        wc_notify --data-binary '{"status": "SUCCESS"}'
  ...
```

Heat vježba 5

Heat vježba 5 ne može se vršiti unutar DevStack okruženja. Vježbu je potrebno izvršiti unutar klasične OpenStack distribucije.

Opis skripte 5:

Skripta 5 bazirana je na Heat vježbi 4. Demonstrira način na koji možemo automatizirati deployment web usluga te njihovo povezivanje. Web servisi su pisani pomoću PHP-a te za međusobno korištenje primjenjuju WSDL.

Ovaj primjer prikazuje pokretanje dvije Instance. Service instanca će pogoniti web servis dok će Client instanca putem klijente aplikacije dohvaćati WSDL datoteku sa Service instance te putem nje koristiti metode koje pruža web servis. Instalacija LAMP aplikacije automatski se izvršava putem apt-get naredbe tijekom Cloud-init procesa kao demonstrirano u Heat vježbi 4. LAMP koristimo za pokretati PHP aplikacije koje ćemo skinuti sa odgovarajućeg git repozitorija.

Uz korištenje WSDL-a i php aplikacija, ova Heat skripta također automatski kreira floating IP adrese te ih veže za kreirane Service i Client instance. Tijekom uspostave web servisa koristimo floating IP adrese kako bi se osigurali da će servisi biti međusobno vidljivi i u slučaju da nisu pokrenuti unutar iste privatne virtualne mreže.

Resurs korišten pri kreaciji nove floating IP adrese je **OS::Neutron::FloatingIP** tipa. Putem svojstva `floating_network` potrebno je definirati javnu mrežu unutar koje će biti rezervirana floating IP adresa. Resurs tipa **OS::Neutron::FloatingIPAssociation** korišten je logičko povezivanje kreirane floating IP adrese i mrežnog sučelja instance. Svojstva su mu `floatingip_id` i `port_id` putem kojih stvara vezu.

Dio service.yaml skripte koju skripta 5 poziva:

```
resources:
  ...
  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: { get_param: public_network }

  floating_ip_assoc:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: floating_ip }
      port_id: { get_resource: port }
  ...
```

4 Python skripte

Iduće vježbe demonstrirati će način korištenja OpenStack API-ja putem python skripti. Svaki program koji sačinjava OpenStack ima svoj specifični API putem kojega prima i prikazuje informacije vezane za njegov rad. Lista API adresa svih OpenStack elemenata može se vidjeti na Horizon sučelju preko **Compute \ Access & Security** sučelja pod **API Access** tabom. Ovdje svaka OpenStack usluga ima vlastitu http ili https adresu putem koje joj se može pristupiti. Usluge su nazvane prema vrsti usluge koje pružaju a ne prema nazivima programa koji pružaju te usluge. Ukoliko želimo doznati api Nova usluge potrebno je pronaći adresu za uslugu Compute. Neutron je Networking, Glance je Image itd.

Za razliku od Heat vježbi, većina objašnjenja za način rada python skripti biti će komentirana direktno u kodu.

Autentifikacija

Kako bi python skripte mogle koristiti OpenStack API-je potrebno je izvršiti autentifikaciju. Prije pokretanja same python skripte korisnik mora izvršiti source naredbu opisanu u poglavlju Pristup CLI te sustavu priložiti potrebne podatke. Svaka python skripta, kako bi mogla izvršiti naredbe pojedinih OpenStack usluga, mora instancirati objekt vezan za tu uslugu. Taj objekt će, prilikom instanciranja, iz environment varijabli preuzeti podatke o korisniku te putem njih imati pristup naredbama. Primjer koda za kreaciju keystone objekta je:

```
from os import environ as env
import keystoneclient.v2_0.client as ksclient
keystone = ksclient.Client(auth_url=env['OS_AUTH_URL'],
                           username=env['OS_USERNAME'],
                           password=env['OS_PASSWORD'],
                           tenant_name=env['OS_TENANT_NAME'],
                           region_name=env['OS_REGION_NAME'])
```

Naredbe vezane za keystone element OpenStack-a sada možemo pozivati kao metode keystone objekta. Sintaksa za instanciranje OpenStack usluga je ista u većini slučajeva tako da ćemo stvoriti credentials.py skriptu koju ćemo koristiti za autentifikacije. Skripta se nalazi u mapi "resources" koja se nalazi unutar iste mape kao i skripte koje je pozivaju.

Skripta credentials:

```
#!/usr/bin/env python

from os import environ as env

def get_creds():
    d = {}
    d['auth_url'] = env['OS_AUTH_URL']
    d['username'] = env['OS_USERNAME']
    d['password'] = env['OS_PASSWORD']
    d['tenant_name'] = env['OS_TENANT_NAME']
    d['region_name'] = env['OS_REGION_NAME']
    return d

def get_nova_creds():
    d = {}
    d['auth_url'] = env['OS_AUTH_URL']
    d['username'] = env['OS_USERNAME']
    d['api_key'] = env['OS_PASSWORD']
    d['project_id'] = env['OS_TENANT_NAME']
    d['region_name'] = env['OS_REGION_NAME']
    return d
```


Python vježba 1

Ova skripta koristiti će Glance uslugu kako bi kreirala OS sliku. Prije pokretanja skripte potrebno je preuzeti OS sliku putem naredbe:

```
wget https://cloud-images.ubuntu.com/wily/current/wily-server-cloudimg-amd64-disk1.img
```

Skripta 1:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# dohvaćanje vanjskih funkcija
from os import environ as env
import keystoneclient.v2_0.client as ksclient
import glanceclient.v2.client as glclient
from credentials import get_creds

# instanciranje keystone objekta putem environment varijabli
keystone = ksclient.Client(**get_creds())

# dohvaćanje URL adrese glance API-ja
glance_endpoint = keystone.service_catalog.url_for(service_type='image')

# instanciranje glance objekta putem keystone usluga
glance = glclient.Client(glance_endpoint, token=keystone.auth_token)

# korištenje glance metode za kreaciju elementa OS slike
image = glance.images.create(name="ubuntu_cloud15", visibility="public",
                             disk_format="qcow2",
                             container_format="bare")

# korištenje glance metode za dodavanje OS slike prethodno kreiranom elementu
glance.images.upload(image.id, open('wily-server-cloudimg-amd64-disk1.img',
'rb'))

print "Image Created"
```

Iz skripte 1 vidljivo je da ne moramo znati API adresu Glance usluge već će ona biti preuzeta putem keystone service_catalog metode. Autentifikaciju prema Glance usluzi također vršimo putem keystonea pomoću autentifikacijskog tokena generiranog na temelju naše keystone autentifikacije. Putem images.create metode OpenStack-u šaljemo naredbu za stvaranjem nove slike sa svim potrebnim parametrima.

Python vježba 2

Skripta 2 izlistava sve slike koje se nalaze unutar OpenStack sustava. Izlistava ih po imenu i po veličini. Naknadno od korisnika zahtjeva unos imena te na temelju zadanog imena povlači ID navedene slike.

Skripta 2:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from os import environ as env
import keystoneclient.v2_0.client as ksclient
import glanceclient.v2.client as glclient
import novaclient.client
from credentials import get_creds, get_nova_creds

keystone = ksclient.Client(**get_creds())

glance_endpoint = keystone.service_catalog.url_for(service_type='image')
glance = glclient.Client(glance_endpoint, token=keystone.auth_token)

# instanciranje nova objekta, argument "2" se odnosi na verziju klase
# koja će biti korištena za stvaranje objekta
nova = novaclient.client.Client("2", **get_nova_creds())

# ispis teksta na terminal
print "List of all images by name and size:"

# dohvaćanje liste koja opisuje sve pohranjene slike unutar glance usluge
images = glance.images.list()

# for petlja koja se izvodi za svaki element (image) unutar liste (images)
for image in images:

    # Ispis atributa imena i velicine pojedine pohranjene OS slike
    print("\n%s\n%s" % (image[u'name'], image[u'size']))

# traženje unosa preko terminala od strane korisnika
name = raw_input('\nSearch for image by name: ')

print('\nLooking for %s...\n' % name)

# početak try bloka
try:
    #dohvaćanje specificne OS slike putem njezinog imena
    image = nova.images.find(name=name)
    print('Image found, id is:%s' % image.id)

# izvršava se ukoliko dođe do greške unutar try bloka
except:
    print "Image Not Found"
```

Glance usluga ne podržava pretragu arhive slike po imenu tako da je tu potrebno koristiti Nova uslugu koja implementira funkciju `images.find` kako bi dohvatila relevantni image objekt. Putem Nova usluge možemo po imenu dohvatiti veliki broj objekata vezanih za OpenStack kao što su sigurnosne grupe (`nova.security_groups.find`), flavour-i (`nova.flavors.find`), sigurnosni ključevi (`nova.keypairs.find`), instance (`nova.servers.find`) itd. Korištenjem identifikacijskog broja dobivenog image objekta koristimo `images.update` metodu Glance usluge kako bi promijenili ime postojeće slike.

Python vježba 3

Skripta 3 koristi Neutron uslugu kako bi kreirala novu mrežu zajedno sa pod-mrežom.

Skripta 3:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from os import environ as env
import novaclient.client
from neutronclient.v2_0 import client as neutronclient
from credentials import get_creds, get_nova_creds

nova = novaclient.client.Client("2", **get_nova_creds())

# instanciranje neutron objekta
neutron = neutronclient.Client(**get_creds())

network_name = 'my_net2'

try:
    # pisanje zahtjeva za stvaranjem mreže
    body_net = {'network': {'name': network_name,
                             'admin_state_up': True}}

    # kreacija mreže te pohrana podataka o toj mreži
    netw = neutron.create_network(body=body_net)

    # dohvaćanje ID kreirane mreže
    net_dict = netw['network']
    network_id = net_dict['id']

    print('Network %s created' % network_id)

    # pisanje zahtjeva za stvaranjem pod-mreže
    body_subnet = {'subnets': [{'name': 'my_subnet1',
                                   'cidr': '10.20.1.0/24',
                                   'ip_version': 4,
                                   'dns_nameservers': ['8.8.4.4', '8.8.8.8'],
                                   'network_id': network_id}]}

    # kreacija pod-mreže te pohrana podataka o toj mreži
    subnet = neutron.create_subnet(body=body_subnet)

    print('\nCreated subnet %s\n' % subnet)

# izvršava se nakon što se try blok izvrši bez greške
finally:
    print("Execution completed")
```

Python vježba 4

Skripta 4 stvara router koji povezuje privatnu mrežu kreiranu u vježbi 3 i javnu mrežu.

Skripta 4:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from os import environ as env
import novaclient.client
from neutronclient.v2_0 import client as neutronclient
from credentials import get_creds, get_nova_creds

nova = novaclient.client.Client("2", **get_nova_creds())

neutron = neutronclient.Client(**get_creds())

# korištenje find metode nova objekta za dohvat ID-a privatne i javne mreže
network_id = nova.networks.find(label='my_net2').id
public_network_id = nova.networks.find(label='public').id

# pisanje zahtjeva za stvaranjem routera
body_router = {'router': {'name': 'my_router2',
                          'admin_state_up': True}}

router = neutron.create_router(body=body_router)
router_id = router['router']['id']

# pisanje zahtjeva za stvaranjem porta
# koji na privatnoj mreži sa adresom 10.20.1.1 (gateway)
body_port = {'port': {
    'admin_state_up': True,
    'network_id': network_id,
    'fixed_ips': [{"ip_address": "10.20.1.1"}]
}}

port = neutron.create_port(body=body_port)
port_id = port['port']['id']

# stvaranje gateway-a putem kojega će router imati pristup javnoj mreži
neutron.add_gateway_router(router=router_id, body={"network_id":
public_network_id})

# stvaranje interface-a putem kojega će router imati pristup privatnoj mreži
neutron.add_interface_router(router=router_id, body={"port_id": port_id})

# ispis podataka o stvorenom router-u
router = neutron.show_router(router_id)
print(router)
print("\nExecution Completed\n")
```

Python vježba 5

Skripta 5 stvara instancu na temelju resursa koje smo stvorili u prethodnim vježbama.

Skripta 5:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time
from os import environ as env
import novaclient.client
from credentials import get_nova_creds

nova = novaclient.client.Client("2", **get_nova_creds())

try:

    # odabir resursa za stvaranje instance
    image = nova.images.find(name="ubuntu_cloud15")
    flavor = nova.flavors.find(name="m1.small")
    net = nova.networks.find(label="my_net2")
    nics = [{'net-id': net.id}]

    # stvaranje instance
    instance = nova.servers.create(name="my_inst2", image=image,
                                    flavor=flavor, key_name="my_key1",
                                    nics=nics)

    # čekanje 5 sec. prije ispisa
    # kako bi se nareda unutar nove stigle izvršiti
    print("Sleeping for 5s after create command")
    time.sleep(5)

    # ispis svih instanci
    print("List of VMs")
    print(nova.servers.list())

finally:
    print("Execution Completed")
```

Izvori

<http://openstack-cloud-mylearning.blogspot.hr/2015/02/openstack-juno-devstack-installation.html>

<http://docs.openstack.org/developer/devstack/#all-in-one-single-vm>

<https://developer.rackspace.com/blog/openstack-orchestration-in-depth-part-1-introduction-to-heat/>

<https://developer.rackspace.com/blog/openstack-orchestration-in-depth-part-2-single-instance-deployments/>

<https://developer.rackspace.com/blog/openstack-orchestration-in-depth-part-3-multi-instance-deployments/>

http://docs.openstack.org/developer/heat/template_guide/

<http://superuser.openstack.org/articles/simple-auto-scaling-environment-with-heat>

<https://github.com/rackerlabs/heat-tutorial/tree/master/200.A-Real-Stack>

<http://docs.openstack.org/user-guide/sdk.html>

<http://www.ibm.com/developerworks/cloud/library/cl-openstack-pythonapis/>