Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andrej Gregorka

# Mobilna aplikacija za administracijo zasebnega oblaka OpenStack

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana 2014

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

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: 00150 / 2013
Datum: 14.9.2013

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

Kandidat: **ANDREJ GREGORKA**

Naslov: **MOBILNA APLIKACIJA ZA ADMINISTRACIJO ZASEBNEGA OBLAKA OPENSTACK**
**MOBILE APPLICATION FOR OPENSTACK PRIVATE CLOUD ADMINISTRATION**

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Predstavite koncepte računalništva v oblaku. Preglejte najpomembnejše sisteme za nadzor in upravljanje oblaka oz. oblačne infrastrukture. Podrobno analizirajte platformo OpenStack. Zasnujte, pripravite načrt in izdelajte mobilno aplikacijo za administracijo oblaka na platformi OpenStack. Vašo rešitev primerjajte s sorodnimi in vzpostavite testno okolje.

Mentor: Dekan:

prof. dr. Branko Matjaž Jurič prof. dr. Nikolaj Zimic

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andrej Gregorka, z vpisno številko **63000171**, sem avtor diplomskega dela z naslovom:

*Mobilna aplikacija za administracijo zasebnega oblaka OpenStack*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža B. Juriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter 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 19. marca 2014 Podpis avtorja:

# Kazalo

# Slike

# Povzetek

Osrednja naloga te diplomske naloge je bila izdelava mobilne aplikacije za administracijo oblaka OpenStack za operacijski sistem Android. Poleg administracije pa aplikacija omogoča tudi sprejemanje obvestil o dogodkih v oblaku preko sistema za sporočanje Google Cloud Messaging.

Na začetku diplomske naloge je opisanih nekaj osnovnih pojmov o računalništvu v oblaku s predstavitvijo najbolj razširjenih komercialnih oblačnih platform. Nadalje je prestavljen oblak OpenStack ter predvsem načini za njegovo administracijo, in sicer skupaj s primeri.

Na koncu je predstavljena še aplikacija OpenStackMobile, ki je bila izdelana, z bistvenimi izvlečki iz kode. Podrobneje je predstavljen sistem za sporočanje skupaj z dopolnitvami, ki so bile potrebne na strežniku OpenStack.

## Ključne besede

Oblak, OpenStack, administracija oblaka, mobilne aplikacije, Android, Google Cloud Messaging

# Abstract

The main purpose of this thesis was to create a mobile application for OpenStack cloud administration. Besides administration the application is also capable of receiving messages about cloud events through Google Cloud Messaging system.

In the beginning of this thesis we first go through some basic cloud computing concepts and get to know the most prominent commercial cloud computing platforms. Then the OpenStack cloud is represented, with emphasis on its administration together with examples.

In the end we present OpenStackMobile application that we created, along with its significant code fragments. This is followed by closely examined messaging system and the enhancements made on OpenStack server.

## Keywords

Cloud, OpenStack, Cloud administration, Mobile applications, Android, Google Cloud Messaging

# Poglavje 1

# Uvod

Računalništvo v oblaku postaja zadnja leta čedalje bolj razširjeno tako med posamezniki kot podjetji. K temu sta najbolj pripomogla razvoj širokopasovnih spletnih povezav in razvoj tehnologij, kot je virtualizacija, ki omogočajo računalništvo v oblaku. Vedno več podjetij svojo infrastrukturo seli v oblak, bodisi v javne ali zasebne oblake. Za zasebne oblake je mogoče kupiti več različnih komercialnih rešitev, kot sta na primer VMWare vCloud ali Microsoft Private Cloud, lahko pa izberemo tudi prostodostopno odprtokodno rešitev, od katerih velja za industrijski standard OpenStack, ki si ga bomo podrobneje pogledali tudi v tem diplomskem delu.

Poudarek tega diplomskega dela je predvsem na administraciji oblaka, in sicer na administraciji preko mobilnih naprav. Do določene mere je mogoče oblake preko mobilnih naprav nadzirati že preko njihovih spletnih vmesnikov, vendar ti niso prirejeni za majhne zaslone in nimajo sistemov za obveščanje. V ta namen je bila izdelana mobilna aplikacija OpenStackMobile, ki omogoča administracijo oblaka OpenStack na mobilnem operacijskem sistemu Android. Operacijski sistem Android je bil izbran, ker gre trenutno za najbolj razširjen mobilni operacijski sistem, poleg tega pa so vsa razvojna orodja zanj prosto dostopna in odprtokodna - podobno kot sam oblak OpenStack.

Prvo poglavje opisuje nekaj osnovnih pojmov o računalništvu v oblaku in nadzoru oblaka ter kratek pregled nad štirimi precej razširjenimi javnimi in

zasebnimi oblaki: Amazon Web Services, Microsoft Azure, Microsoft Private Cloud ter VMWare vCloud. Naslednje poglavje predstavi oblak OpenStack ter načine za njegovo administracijo, ki so: spletni vmesnik, ukazna vrstica ter spletni servisi REST. Zadnje poglavje zajema izdelavo mobilne aplikacije skupaj s pomembnimi izvlečki iz kode. Predstavljena je izdelava same aplikacije Android in dodatkov na strežniku OpenStack, ki omogočajo pošiljanje sporočil o dogodkih v oblaku mobilni aplikaciji preko sistema za sporočanje Google Cloud Messaging.

# Poglavje 2

# Računalništvo v oblaku

Računalništvo v oblaku (RvO) lahko definiramo kot računalništvo, v katerem so elastični in skalabilni informacijski viri dostavljeni zunanjim uporabnikom kot servisi s pomočjo internetnih tehnologij [1].

Preden definiramo, kaj oblak sploh je, si poglejmo še nekaj osnovnih pojmov.

## 2.1 Predstavitev osnovnih pojmov

### 2.1.1 Informacijski vir

Informacijski vir (angl. IT resource) nam predstavlja element znotraj oblaka, ki je bodisi fizični ali navidezni. Primeri informacijskih virov so:

- fizični strežnik,
- navidezni strežnik,
- mrežna naprava,
- podatkovna baza,
- programska oprema in
- servis, ki se izvaja na strežniku.

### 2.1.2 Skalabilnost

Skalabilnost (angl. scaling) predstavlja zmožnost informacijskega vira, da povečuje ali zmanjšuje svoje zmogljivosti glede na zahteve.

Poznamo vodoravno in navpično skalabilnost.

Vodoravna skalabilnost pomeni povečevanje števila informacijskih virov enakega tipa. Primer je povečevanje števila instanc navideznega strežnika na fizičnih strežnikih.

O navpični skalabilnosti govorimo, ko zamenjamo obstoječ informacijski vir s takim, ki ima večjo ali manjšo zmogljivost. Primer tega je, ko na fizičnem strežniku zamenjamo obstoječi procesor z zmogljivejšim.

V oblakih je pogostejša vodoravna skalabilnost, pri kateri je čas izpada manjši oziroma ga ni, za razliko od navpične skalabilnosti, pri kateri je zaradi menjave komponent lahko čas izpada daljši.

### 2.1.3 Oblak

Oblak je skupek informacijskih virov, ki jih je mogoče na daljavo upravljati, meriti in so skalabilni [1].

Večini oblakov so skupne naslednje lastnosti:

- Uporaba na zahtevo - Uporabniki oblaka si lahko sami vzpostavijo okolje in informacijske vire, ki jih potrebujejo.
- Dostop iz različnih naprav - Storitve informacijskih virov oblaku so ves čas dostopne različnim napravam, običajno preko programskih vmesnikov (API).
- So večnajemniški - Storitve v oblaku hkrati uporablja več uporabnikov (najemnikov), ki so med seboj izolirani. Oblak samodejno razporeja vire med najemniki glede na potrebe.
- Elastičnost - Oblak povečuje oziroma zmanjšuje svoje zmogljivosti (skalabilnost) glede na potrebe. Oblačni ponudniki z več razpoložljivimi informacijskimi viri ponujajo večjo elastičnost.

- Merjenje porabe - Oblak vsebuje servise za meritve porabe informacijskih virov. Na podlagi teh meritev lastnik oblaka zaračuna uporabniku storitev za tisti čas, ko jo je uporabljal, oziroma toliko enot, kot jih je dejansko porabil.

- Odpornost - V oblaku imamo na voljo odvečne informacijske vire, ki jih lahko uporabimo v primeru odpovedi obstoječih virov ali pri povečanem številu zahtev. Dodatni viri so lahko razporejeni tudi na različnih fizičnih lokacijah, kar poveča odpornost oblaka.

### 2.1.4 Modeli oblačnih storitev

Različni modeli oblačnih storitev predstavljajo kombinacije informacijskih virov, ki jih ponudniki oblačnih storitev ponujajo svojim uporabnikom. Najbolj pogosti so [1, 25]:

- infrastruktura kot storitev (IaaS),
- platforma kot storitev (PaaS) in
- programska oprema kot storitev (SaaS).

#### Infrastruktura kot storitev

Infrastruktura kot storitev (angl. Infrastucture-as-a-Service - IaaS) predstavlja sklop surovih informacijskih virov, kot so navidezni strežniki, navidezna omrežja in ostala strojna oprema, do katere lahko uporabniki dostopajo preko oblaka. Za razliko od klasičnih ponudnikov gostovanja je tu strojna oprema virtualizirana in skalabilna.

V tem diplomskem delu se večinoma ukvarjamo z infrastrukturo kot storitvijo. Tako vrsto oblaka lahko zgradimo tudi s projektom OpenStack, ki je opisan v poglavju 3.1.

### Platforma kot storitev

Pri platformi kot storitvi (angl. Platform-as-a-Service - Paas) gremo nivo višje in poleg strojne opreme uporabniku ponudimo tudi platformo, na kateri lahko zgradi oziroma poganja svoje rešitve. Primer platforme kot storitve je aplikacijski in podatkovni strežnik v oblaku.

### Programska oprema kot storitev

Programska oprema kot storitev (angl. Software-as-a-Service - SaaS) gre še nivo višje in ponuja programsko opremo v oblaku končnim uporabnikom. Tipičen primer tega je elektronska pošta v oblaku (Gmail.com, Yahoo Mail, Outlook.com itd.).

Uporabljamo lahko različne kombinacije modelov. Spodaj imamo tipično IaaS, na kateri zgradimo PaaS in na platformi aplikacijo (SaaS), kot prikazuje slika 2.1.


Slika 2.1: Plasti modelov oblačnih storitev.

### 2.1.5 Pomembne tehnologije

Oblaki temeljijo na vrsti tehnologij, ki omogočajo njihovo izgradnjo in uporabo. Bistvene tehnologije so:

- internet in širokopasovna omrežja,
- podatkovni centri,
- virtualizacija,
- spletne tehnologije,
- večnajemniške tehnologije in
- spletni servisi.

#### Internet in širokopasovna omrežja

Vsi oblaki morajo biti povezani v omrežje. Javni oblaki so vedno povezani tudi v internet. Z razvojem širokopasovnih internetnih povezav se je tako povečal tudi krog potencialnih uporabnikov javnih oblakov. Zasebni oblaki so lahko dostopni tudi samo v lokalnih omrežjih, čeprav je tudi večina teh še vedno dostopna tudi preko interneta z upoštevanjem ustreznih varnostnih mehanizmov.

#### Podatkovni centri

Večina fizične infrastrukture, kot so fizični strežniki, omrežja in enote za shranjevanje podatkov, je v specializiranih zgradbah, ki jih imenujemo podatkovni centri.

Podatkovni centri so zgrajeni tako, da so stroški vzdrževanja čim nižji, da jih je enostavno vzdrževati ter da je zagotovljena ustrezna varnost. Elementi podatkovnih centrov so:

- kompaktno zloženi fizični strežniki (rackmount),
- diskovna polja,

- omrežne povezave in
- pomožni sistemi (hlajenje, napajanje, prezračevanje, varnostni sistemi).

### Virtualizacija

Virtualizacija je proces, kjer fizični informacijski vir pretvorimo v enega ali več navideznih (logičnih) informacijskih virov. Informacijski viri, ki jih lahko virtualiziramo, so tipično strežniki, podatkovne shrambe in omrežja.

Programsko opremo, ki omogoča virtualizacijo strežnikov, imenujemo hipervizor. Hipervizor omogoča preslikavo fizičnega strežnika v več navideznih (logičnih) strežnikov.

Fizični strežnik imenujemo tudi gostitelj. Hipervizor se lahko izvaja neposredno na gostitelju, v tem primeru govorimo o virtualizaciji na nivoju strojne opreme. Lahko pa se hipervizor izvaja tudi kot program znotraj operacijskega sistema gostitelja. V tem primeru govorimo o virtualizaciji na nivoju operacijskega sistema.

### Spletne tehnologije

Osnovne spletne tehnologije, ki se uporabljajo pri izgradnji oblakov, so:

- URL - enolični krajevnik vira (angl. Universal Resource Locator) - Standardna sintaksa, s katero določimo lokacije različnih virov v oblaku.
- HTTP - Primarni protokol za prenos informacij preko spleta.
- Označevalni jeziki, kot sta HTML in XML - HTML se običajno uporablja za prezentacijo, XML pa za prenos podatkov.

### Večnajemniške tehnologije

Večnajemniške tehnologije omogočajo, da več uporabnikov (najemnikov) hkrati uporablja informacijske vire, vendar se ne zavedajo medsebojnega obstoja. Osnovne lastnosti so:

- Izolacija - Akcije posameznega uporabnika ne vplivajo na druge in se jih ti ne zavedajo. Izolirani so tudi podatki uporabnikov.
- Varnost podatkov - Uporabnik lahko dostopa samo do svojih podatkov.
- Obnovitev - Izdelava varnostnih kopij in obnovitev podatkov se izvajata ločeno za vsakega najemnika posebej.
- Skalabilnost - Oblačna storitev lahko povečuje ali zmanjšuje svoje zmogljivosti in se tako prilagaja povečanim potrebam najemnika ali povečanemu številu najemnikov.
- Merjenje porabe - Za vsakega najemnika se poraba meri in zaračunava ločeno.

#### Spletni servisi

Poznamo 2 tipa spletnih servisov:

- SOAP - Protokol za spletne servise, ki temelji na XML-tehnologiji. Metode so definirane s pomočjo jezika WSDL.
- REST - Za razliko od SOAP nimajo specificiranih vmesnikov. Tipično so implementirane preko HTTP-metod (GET, POST, PUT, DELETE).

### 2.1.6 Načini postavitve oblaka

Poznamo štiri osnovne načine postavitve oblaka glede na lokacijo fizičnih informacijskih virov [1, 25]:

- javni oblak,
- zasebni oblak,
- hibridni oblak in
- oblak skupnosti.

### Javni oblak

O javnem oblaku govorimo takrat, ko ima informacijske vire v lasti tretja oseba, ki ponuja oblačne storitve, in ne organizacija, ki storitve uporablja. Organizacijo, ki storitve ponuja, imenujemo ponudnik oblačnih storitev. Ta svoje oblačne storitve, ki so lahko katerikoli od modelov, opisanih v poglavju 2.1.4, ponuja na trgu in so običajno dostopne preko interneta. Tipična primera javnega oblaka sta Amazon Web Services in Azure.

### Zasebni oblak

Zasebni oblak ima za razliko od javnega v lasti organizacija, ki njegove storitve tudi uporablja. Tehnično gledano, razlika v arhitekturi med javnim in zasebnim oblakom ni velika, razlika je predvsem v varnosti. Fizični informacijski viri so v zasebnem oblaku na lokaciji (ali več lokacijah) organizacije, ki oblak tudi uporablja in niso deljeni z drugimi organizacijami.

### Hibridni oblak

O hibridnem oblaku govorimo takrat, ko organizacija uporablja tako javni kot zasebni oblak. Deljenje informacijskih virov med javnim in zasebnim oblakom lahko poteka na dva načina. V prvem določene rešitve tečejo na javnem, določene pa na zasebnem oblaku. Tipično organizacije uporabljajo zasebne oblake za varnostno bolj občutljive podatke, na javnih pa so postavljene aplikacije, ki varnostno niso občutljive. Drugi primer hibridnega oblaka pa je aplikacija, ki običajno teče na zasebnem oblaku, ko pa se zaradi povečanega števila zahtev pojavi potreba po dodatnih virih, se ti aktivirajo v javnem oblaku. V tem primeru govorimo o t. i. *cloudburstingu*.

### Oblak skupnosti

Oblak skupnosti je podoben javnemu oblaku, s to razliko, da njegovih storitev ne more najeti katerakoli organizacija, ampak je dostop namenjen samo določenim organizacijam s skupnimi cilji oziroma zahtevami (npr. varnostna

politika). Oblak skupnosti imajo lahko v lasti organizacije, ki ga uporabljajo, ali pa ga zanje upravlja ponudnik oblačnih storitev.

## 2.2 Nadzor oblaka in infrastrukture

Informacijske vire v oblaku je treba pred uporabo postaviti in konfigurirati, kasneje pa vzdrževati [1]. Večina oblačnih sistemov ima mehanizem za administracijo sistema na daljavo. Običajno je ta implementiran v obliki nadzorne konzole, ki je dostopna kot spletna stran ali odjemalec za različne operacijske sisteme. Dodatno pa je na voljo še programski vmesnik (API), preko katerega lahko uporabniki implementirajo svoje rešitve za administracijo.

Preko nadzorne konzole lahko uporabniki in nadzorniki običajno izvajajo naslednja opravila:

- postavitev in konfiguracija informacijskih virov v oblaku,
- zaganjanje in ustavljanje informacijskih virov glede na potrebe,
- nadzor delovanja, uporabe in zmogljivosti,
- pregled stroškov,
- pregled in nadzor uporabniških računov, uporabniških vlog in pravic dostopa,
- pregled dnevnikov dostopa do storitev in
- planiranje kapacitet.

### 2.2.1 Prednosti in slabosti računalništva v oblaku

#### Prednosti RvO

Glavne prednosti računalništva v oblaku s poslovnega stališča so:

- Lažje planiranje kapacitet - Virtualizacija nam omogoča razporejanje informacijskih virov glede na potrebe med različnimi uporabniki v organizaciji. Tako je planiranje olajšano, saj lahko potrebne kapacitete planiramo na višjem nivoju.

- Nižji stroški - Razporejanje in najem informacijskih virov glede na potrebne ter pri javnih oblakih tudi nižja začetna cena postavitve znižajo strošek postavitve in vzdrževanja računalniške infrastrukture v organizaciji.

- Večja agilnost organizacije - Lahko se hitreje prilagajamo spremembam zaradi skalabilnih informacijskih virov v oblaku.

**Slabosti RvO**

- Varnost - Dostopnost javnih oblakov in deljenje informacijskih virov med večimi odjemalci oblačnih storitev lahko povečata varnostno tveganje in možnosti za vdor v sistem.

- Zasebnost - Ponudnik oblačnih storitev ima fizični dostop do podatkov organizacij, ki njegove storitve uporabljajo, zato obstaja možnost uhajanja informacij in posredovanje teh tretjim osebam.

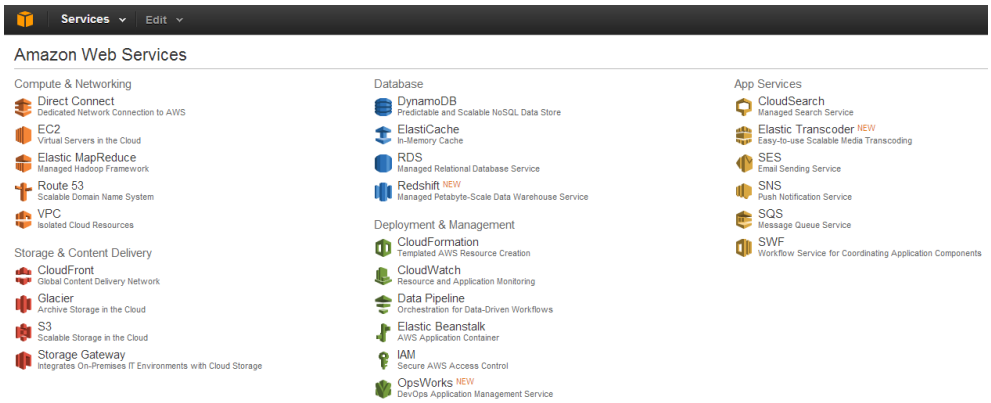
- Skladnost z zakonodajo - Zakonodaja nekaterih držav postavlja omejitve glede lokacije hranjenja določenih podatkov, zato je lahko računalništvo v oblaku neprimerno za določene aplikacije.

## 2.3 Pregled komercialnih sistemov za nadzor oblaka

V spodnjih poglavjih si bomo ogledali štiri najbolj pogoste komercialne oblake in njihove mehanizme za administracijo.

### 2.3.1 Amazon Web Services

Amazon Web Services (v nadaljevanju AWS) je zbirka več kot 20 oblačnih storitev podjetja Amazon, ki nam omogočajo postavitev infrastrukture v oblaku (IaaS). Najbolj popularni storitvi sta Amazon EC2, ki jo lahko uporabimo za postavitev virtualnih instanc v oblaku, in Amazon S3 za shranjevanje objektov v oblaku. Slika 2.2 prikazuje celoten seznam spletnih servisov, ki so na voljo v nadzorni plošči sistema AWS.




Slika 2.2: Nadzorna plošča oblaka AWS.

#### Amazon EC2

Amazon EC2 ali Elastic Compute Cloud omogoča najem virtualnih strežnikov različnih zmogljivosti v oblaku [3]. Uporaba virtualnih strežnikov se plačuje za vsako uro uporabe posebej, lahko pa se najamejo tudi vnaprej. Kot hipervizor se uporablja Xen, na virtualnih strežnikih pa so na voljo različni operacijski sistemi od Linux, BSD do Windows Server. Izbiramo lahko različne fizične lokacije virtualnih strežnikov.

#### Amazon S3

Amazon S3 ali Simple Storage Service omogoča objektno shranjevanje podatkov v oblaku [4]. Na voljo imamo operacije za branje, pisanje in brisanje

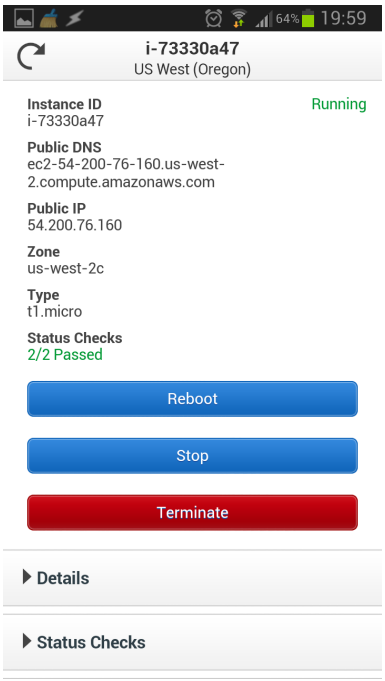
objektov, velikosti od enega bajta to petih terabajtov. Dostop je mogoč preko spletnih servisov REST ali SOAP. Omogočeno je prenašanje podatkov v šifrirani obliki. Podatki so lahko shranjeni na različnih fizičnih lokacijah. Zaračunava se število porabljenih GB na mesec ter število operacij, ki jih izvajamo (branje, pisanje, brisanje).

### Administracija AWS

Amazon spletne servise lahko administriramo preko spletnega vmesnika, kot je bilo prikazano na sliki 2.2.

Poleg spletnega vmesnika je za nekatere izmed oblačnih storitev (npr. EC2) na voljo še administracija preko ukazne vrstice. Za večino storitev pa je na voljo tudi programski vmesnik REST oziroma SOAP.

Za administracijo nekaterih oblačnih storitev Amazon, predvsem EC2 in z njim povezanih storitev je na voljo mobilna aplikacija za operacijska sistema Android in iOS. Primer administracije virtualnega strežnika preko mobilne aplikacije prikazuje slika 2.3.




Slika 2.3: Nadzor virtualnega strežnika v oblaku AWS preko mobilne aplikacije.

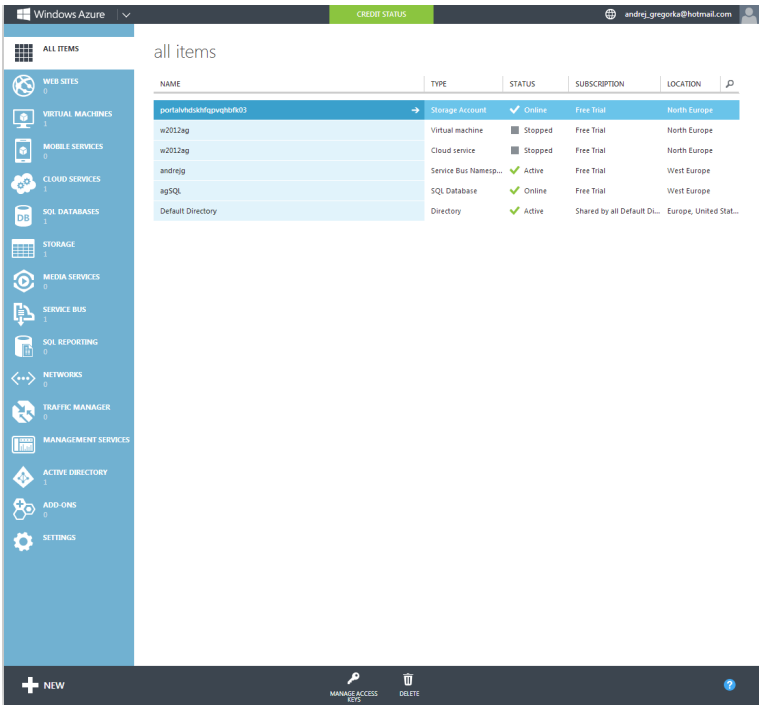
### 2.3.2 Azure

Windows Azure je zbirka oblačnih storitev podjetja Microsoft, ki podobno kot AWS omogočajo postavitev infrastrukture v oblaku (IaaS), dodatno pa so na voljo tudi storitve PaaS, saj imamo denimo možnost najema podatkovnih baz in aplikacijskih strežnikov [26].

Za postavitev infrastrukture v oblaku Azure lahko najamemo virtualne strežnike, ki poganjajo operacijski sistem Windows oziroma Linux. Za virtualizacijo se uporablja hipervizor Hyper-V, ki se uporablja tudi v operacijskih sistemih Windows Server. Podobno kot pri AWS se tudi pri oblaku Azure zaračunava ura uporabe. Virtualne strežnike lahko integriramo z ostalimi oblačnimi storitvami, ki jih ponuja Azure, kot so navidezna omrežja in storitve za shranjevanje podatkov ter avtentikacijo (Active Directory).

### Administracija oblaka Azure

Administracija oblaka Azure je mogoča preko spletne nadzorne plošče, ki jo prikazuje slika 2.4. Dodatno je administracija mogoča preko ukazne vrstice, in sicer za orodje PowerShell, v katerega lahko namestimo module za administracijo oblaka Azure. Na voljo so tudi spletni servisi REST.




Slika 2.4: Nadzorna plošča oblaka Azure.

## 2.3.3 Microsoft Private Cloud

Microsoft Private Cloud je zbirka produktov podjetja Microsoft za postavitev zasebnega oblaka [13]. Oblak temelji na operacijskem sistemu Windows Server 2008 R2 oziroma Windows Server 2012 s hipervizorjem Hyper-V. Možnost imamo uporabljati tudi druge hipervizorje (VMWare in Citrix).

Za konfiguracijo in nadzor oblaka se uporablja programska oprema System Center. Oblak je mogoče administrirati tudi iz ukazne vrstice, in sicer v

okolju PowerShell.

### 2.3.4 VMmare vCloud

VMWare vCloud je zbirka produktov podjetja VMWare za postavitev zasebnih oblakov [23]. Osnovana je okoli oblačnega operacijskega sistema vSphere. Kot hipervizor se uporablja VMWare ESX ali ESXi. Poleg tega so v vCloud vključeni še naslednji produkti:

- vCenter Site Recovery Manager - izdelava varnostnih kopij in redundantnih kopij navideznih strežnikov,
- vCloud Networking and Security - orodja za izdelavo navideznih omrežij, požarnih zidov, uravnalnikov bremen in ostale omrežne infrastrukture,
- vCloud Automation Center - avtomatizacija procesov v oblaku,
- vCenter Operations Manager Suite - orodje za pogled v celotno sliko oblaka in delovanje ter zasedenost različnih delov sistema,
- vCloud Director - orodje za orkestracijo in hitro postavitev celotne infrastrukture na podlagi predlog.

Oblak nadziramo preko aplikacije vSphere Client. Na voljo sta tudi spletna različica, do katere se dostopa preko spletnega brskalnika, ter mobilna različica za operacijska sistema iOS in Android. Dodatno imamo na voljo še odjemalca za ukazno vrstico ter možnost dostopa do oblaka preko spletnih servisov SOAP.

# Poglavje 3

# OpenStack

OpenStack [15] je odprtokodna platforma za razvoj javnih in zasebnih oblakov. Projekt je sestavljen iz več medsebojno povezanih komponent, ki skupaj tvorijo celostno oblačno storitev. Nad projektom bdi neprofitna organizacija OpenStack Foundation, ustanovljena septembra 2012. Poleg tega pa je v projekt vključenih tudi več kot 200 velikih podjetij, med drugim Oracle, IBM, Intel, RedHat, Cisco, Dell, Rackspace.

## 3.1 Predstavitev oblaka OpenStack

Projekt OpenStack je sestavljen iz več medsebojno povezanih komponent [9]. Vse komponente so napisane v programskem jeziku Python. Nova različica je v grobem izdana vsakih 6 mesecev. Trenutna izdaja nosi ime Havana in je bila izdana 17. oktobra 2013. Vsa izvorna koda projekta je prosto dostopna pod licenco Apache 2.0.

V grobem lahko komponente OpenStacka razdelimo na tri dele, to so:

- računski (Compute),
- mrežni (Networking) in
- podatkovna shramba (Storage).

Dodatno so na voljo še podporne storitve, in sicer:

- servis za identiteto (Identity Service),
- servis za slike sistemov (Image Service),
- servis za meritve (Telemetry Service) in
- servis za orkestracijo (Orchestration Service).

Dodatna komponenta, ki omogoča lažji nadzor nad celotnim oblakom, je nadzorna plošča OpenStack Dashboard, ki je podrobneje opisana v poglavju 3.2.2.

### 3.1.1 Komponente oblaka OpenStack

#### Računski sistem - Nova

Nova v projektu OpenStack služi kot nadzornik virtualne infrastrukture (angl. Virtual Infrastructure Manager - VIM). Organizacijam in ponudnikom oblačnih storitev omogoča uporabo informacijskih virov na zahtevo. Arhitektura je zastavljena tako, da je asinhrona in vodoravno skalabilna. Podprtih je več različnih hipervizorjev. Tipično se uporabljata KVM ali XenServer, podpira pa tudi komercialne hipervizorje, kot sta VMWare ESXi ali Hyper-V.

Celoten sistem je mogoče upravljati preko ukazne vrstice, vmesnika API ali nadzorne plošče.

Računski sistem Nova je sestavljen iz naslednjih komponent:

- API-vmesnik - Programski vmesnik za upravljanje z računskim sistemom Nova. Temelji na REST-metodah.
- Podatkovna baza - Shramba metapodatkov računskega sistema. Tipično gre za podatkovno bazo SQL. Podprti sta MySQL in PostgreSQL, poleg teh pa še sqlite3, ki pa ni primerna za produkcijska okolja.
- Sporočilna vrsta - Komponenta za prenašanje sporočil med različnimi deli sistema. Običajno se uporablja RabbitMQ, lahko pa tudi kakšna druga implementacija protokola AMQP.

- Jedro sistema - Glavno vlogo igra proces `nova-compute`, ki skrbi za interakcijo s hipervizorjem. Ukaze dobiva iz čakalne vrste, pri tem pa mu pomaga proces `nova-scheduler`. Dostop do podatkovne baze poteka preko procesa `nova-conductor` in ne z neposrednim dostopom do baze. Tipična ukaza, ki ju jedro sistema Nova pošilja hipevizorju, sta ukaz za zagon ali ustavitev navideznega strežnika.

### Mrežni sistem - Neutron

Neutron je sistem, s katerim vzpostavimo "Omrežje kot storitev". Omogoča vzpostavitev in nadzor nad navideznimi omrežji ter IP-naslovi, vključno s plavajočimi IP-naslovi, ki omogočajo dinamično preusmerjanje prometa v času izpadov ali vzdrževanja sistema.

Uporabniki lahko ustvarjajo svoja navidezna omrežja, nadzirajo promet in povezujejo svoje navidezne strežnike v omrežje.

Sistem omogoča tudi izdelavo razširitev, ki lahko segajo od uravnalnikov bremen, požarnih zidov, sistemov za detekcijo vdorov do navideznih zasebnih omrežij.

### Podatkovna shramba - Swift, Cinder

OpenStack ponuja dva načina shranjevanja podatkov - objektno in na nivoju blokov.

Shranjevanje na nivoju blokov je nizkonivojsko shranjevanje podatkov, kjer nimamo datotečnega sistema, ampak bloke, fiksne dolžine, v katere shranjujemo podatke. V sistemu OpenStack za to skrbi projekt Cinder.

Za shranjevanje objektov je zadolžen projekt Swift. Pri shranjevanju objektov shranjujemo podatke v obliki objekta, ki je sestavljen iz vsebine in metapodatkov. Na voljo imamo uporabniški vmesnik (API), s katerim lahko enostavno beremo ali zapisujemo objekte. Gre za podoben način shranjevanja podatkov, kot ga ponuja Amazon S3, opisan v poglavju 2.3.1.

### Servis za identiteto - Keystone

Servis za identiteto predstavlja osrednji imenik uporabnikov oblaka OpenStack. Uporablja se za avtentikacijo uporabnikov v vse ostale dele sistema. Omogočena je tudi integracija z drugimi uporabniškimi imeniki (LDAP), kar organizacijam omogoča enoten sistem za prijavo (angl. Single Sign On - SSO). Administratorjem servis za identiteto omogoča urejanje uporabnikov in projektov v oblaku ter določanje njihovih pravic.

Dodatna storitev servisa je, da omogoča uporabnikom izpis vseh servisov (in njihovih URL-naslovov), do katerih imajo omogočen dostop.

### Servis za slike sistemov - Glance

Servis za slike sistemov omogoča uporabnikom izdelavo predlog sistemov, na podlagi katerih lahko hitro postavijo nove instance. Omogoča tudi deljenje slik med uporabniki, ki lahko izbirajo med obstoječimi slikami sistemov ali ustvarijo svoje. Podprti so različni formati slik, med drugim tudi VMWare, Hyper-V, VirtualBox in Qemu/KVM.

Na servis je mogoče shraniti tudi posnetek sistema (angl. snapshot), kar omogoča hitro izdelavo varnostnih kopij.

### Servis za meritve - Ceilometer

Servis za meritve se uporablja za merjenje porabe virov in ostalih kazalnikov v oblaku. Administratorjem oblaka s tem omogoča lažji nadzor in pregled nad porabo.

### Servis za orkestracijo - Heat

Gre za servis, ki omogoča izdelavo predlog za različne tipe infrastrukture. Predloge nato uporabimo za hitro postavitev dodatne infrastrukture na svojem oblaku. S predlogami lahko definiramo konfiguracije za računski, mrežni in podatkovni sistem ter tudi vse aktivnosti, ki se izvedejo po postavitvi sistema.

Vgrajena je tudi povezava s servisom za meritve, kar omogoča samodejno povečevanje določenih zmogljivosti glede na potrebe.

### 3.1.2 Prednosti oblaka OpenStack

OpenStack ponuja naslednje prednosti pred konkurenčnimi sistemi za postavitev infrastrukture v oblaku [18]:

- Odprtost - Vsa izvorna koda projekta OpenStack je prosto dostopna. Tako se izognemo zaprtim protokolom in odvisnosti od določenega proizvajalca programske opreme.
- Skalabilnost - OpenStack omogoča visoko skalabilnost, kar dokazujejo številni produkcijski sistemi, katerih velikost shrambe podatkov se meri v petabajtih.
- Industrijski standard - Kot je bilo že omenjeno v uvodu poglavja, so v projekt OpenStack vključena številna velika podjetja, kot so Oracle, IBM, Intel, RedHat, Cisco, Dell, Rackspace ... Med uporabniki pa so tudi številna prepoznavna podjetja in organizacije, kot so NASA, Yahoo, Paypal, CERN, Deutche Telekom ...
- Fleksibilnost - V posameznih delih sistema je podprtih več različnih tehnologij. Računski sistem Nova tako lahko uporablja vrsto različnih hipervizorjev, kot so VMWare ESX, Hyper-V, KVM, QEMU, Xen in drugi. Podobno tudi servis za slike sistemov Glance podpira več različnih formatov slik.

## 3.2 Nadzor oblaka OpenStack

Za nadzor in administracijo imamo na voljo tri možnosti. Prva in najbolj pogosta je ukazna vrstica, preko katere so na voljo vsi deli sistema. Druga

možnost je grafični vmesnik, ki je na voljo kot OpenStack Dashboard oziroma Horizon in je bolj prijazen predvsem za nove uporabnike oblaka. Tretja možnost je OpenStack REST API, ki omogoča dostop do sistema preko HTTP-metod in ga lahko uporabljajo druge aplikacije, ki potrebujejo dostop do oblaka.

### 3.2.1 Ukazna vrstica

Vsak izmed servisov v oblaku OpenStack ima na voljo svojega odjemalca v ukazni vrstici. Odjemalci so napisani v programskem jeziku Python in v ozadju kličejo OpenStack REST API, ki je opisan v poglavju 3.2.3. Na voljo imamo naslednje ukaze:

- `ceilometer` - vmesnik za meritve in telemetrijo,
- `cinder` - upravljanje in pregled servisa za bločno shranjevanje podatkov,
- `glance` - upravljanje slik navideznih strežnikov,
- `heat` - vmesnik za orkestracijo,
- `keystone` - avtentikacija ter upravljanje uporabnikov, projektov in pravic,
- `neutron` - upravljanje navideznih omrežnih sistemov,
- `nova` - upravljanje in pregled navideznih strojev ter njihovih slik in
- `swift` - upravljanje shrambe objektov.

#### Primeri uporabe ukazne vrstice

Spodnji primer prikazuje ukaz, ki nam vrne seznam vseh projektov:

```
keystone --os-username admin --os-password stack
  --os-auth-url http://192.168.0.102:5000/v2.0  tenant-list
```

Kot rezultat dobimo seznam projektov v naslednji obliki:

```
+----------------------------------+--------+---------+
|                id                |  name  | enabled |
+----------------------------------+--------+---------+
| b21cd5a65b9348a4bd9634a847f9fbe9 | admin  |   True  |
| e729ae09b5f749cd9ed4aa3f92cd98ec | andrej |   True  |
| e10bf4935ec5416faf79eabd843335ca |  demo  |   True  |
+----------------------------------+--------+---------+
```

Drugi primer prikazuje ukaz za izpis seznama virtualnih strojev na projektu admin:

```
nova --os-username admin --os-password stack
 --os-tenant-name admin
 --os-auth-url http://192.168.0.102:5000/v2.0  list
```

Dobimo rezultat:

```
+--------------------------------------+---------------+...
| ID                                   | Name          | ...
+--------------------------------------+---------------+...
| b0923416-2a61-46f3-ab97-b939bb4db85d | dns-server-02 | ...
| f6de8a3a-4b68-430e-89e9-abdd98582163 | web-server-01 |...
+--------------------------------------+---------------+...
```

Tretji primer prikazuje ukaz, ki ustavi računalnik z imenom web-server-01:

```
nova --os-username admin --os-password stack
 --os-tenant-name admin
 --os-auth-url http://192.168.0.102:5000/v2.0
 suspend web-server-01
```

### 3.2.2 OpenStack Dashboard

Druga možnost za nadzor oblaka OpenStack je OpenStack Dashboard oziroma Horizon [17]. Horizon je spletna aplikacija, zgrajena na ogrodju Python Django, ki omogoča uporabnikom in administratorjem oblaka nadzor in pregled dogajanja v sistemu.

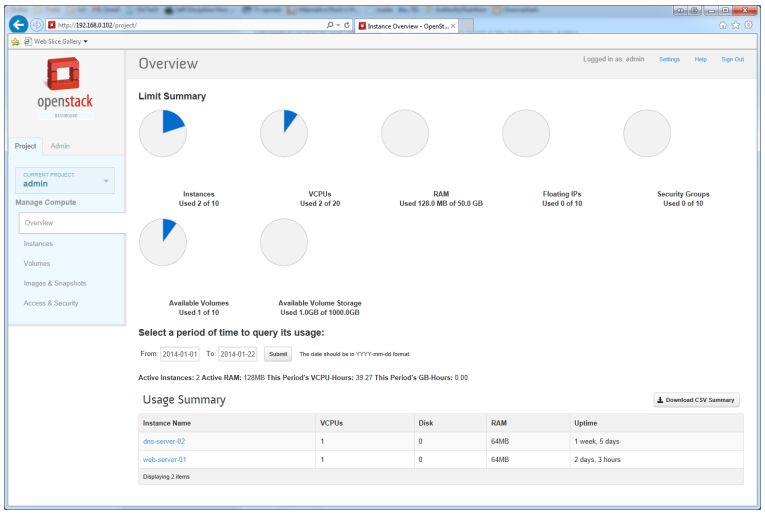
Kot vsi ostali deli sistema OpenStack je tudi Horizon odprtokoden in razširljiv ter omogoča dodajanje novih modulov v aplikacijo, s katerimi lahko podpremo dodatne funkcionalnosti, na primer zaračunavanje storitev.

V grobem je Horizon razdeljen na tri dele:

- Projekti - Modul je namenjen končnim uporabnikom za nadzor instanc, navideznih diskovnih pogonov, shranjenih slik sistema in omrežne infrastrukture.
- Administracija - Namenjena je administratorjem za nadzor informacijskih virov v oblaku, urejanje uporabnikov in projektov ter določanje omejitev.
- Nastavitve - Urejanje osebnih nastavitev uporabnika.

#### Projekti

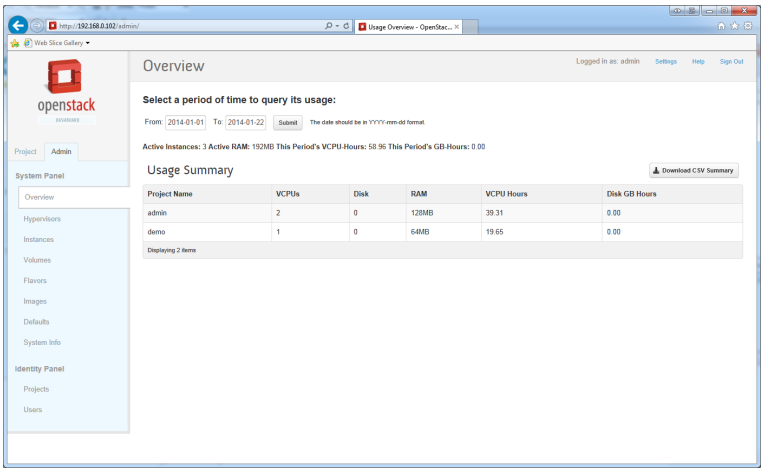
Na strani za projekte si uporabnik lahko izbere enega od projektov, do katerega ima dostop. Za ta projekt se potem prikaže pregled osnovnih podatkov, kot prikazuje slika 3.1. Uporabnik ima na voljo tudi strani za nadzor instanc, navideznih pogonov, slik sistemov in urejanje pravic dostopa do informacijskih virov.




Slika 3.1: Pregled osnovnih podatkov o projektu OpenStack.

### Administracija

Na administratorski strani (slika 3.2) lahko nadzorniki oblaka urejajo in pregledujejo vse instance, navidezne pogone, slike sistemov in omrežno infrastrukturo. Na voljo so tudi strani za urejanje privzetih nastavitev in pregled osnovnih podatkov o oblaku, kot so denimo IP-naslovi posameznih delov sistema. Na voljo je tudi modul za urejanje projektov in uporabnikov oblaka ter dodeljevanje pravic dostopa do projektov uporabnikom.




Slika 3.2: Administratorska nadzorna plošča.

#### Nastavitve

Gre za najenostavnejši del sistema. V njem lahko uporabnik zamenja svoje geslo ter prilagodi nekatere nastavitve prikaza, kot sta jezik in število prikazanih elementov na stran v seznamih.

### 3.2.3 OpenStack API

Tretja možnost za nadzor oblaka OpenStack je REST API [16], ki uporablja HTTP-metode (GET, POST, PUT in DELETE) za administracijo sistema. Podatki se lahko preko teh metod prenašajo v obliki XML ali JSON. V spodnjem primeru in kasneje v opisani aplikaciji je bil uporabljen JSON.

Prva metoda, ki jo je vedno treba klicati, je metoda za prijavo v sistem. Po uspešni prijavi nam sistem vrne žeton, ki ga uporabljamo za avtentikacijo pri vseh ostalih ukazih. Žeton je običajno veljaven 24 ur oziroma odvisno od nastavitve v sistemu.

#### Pregled OpenStack API

OpenStack ponuja dostop do naslednjih storitev preko REST API:

- Compute API - upravljanje in pregled navideznih strežnikov in njihovih slik,
- Block Storage API - upravljanje in pregled servisa za bločno shranjevanje podatkov,
- Identity Service API - avtentikacija ter upravljanje uporabnikov, projektov in pravic,
- Image Service API - upravljanje slik navideznih strežnikov,
- Networking API - upravljanje navideznih omrežnih sistemov,
- Object Storage API - upravljanje shrambe objektov,
- Orchestration API - vmesnik za orkestracijo,
- Telemetry API - vmesnik za meritve.

### Primer uporabe OpenStack API

Kot primer bomo pokazali, kako preko OpenStack REST API izpišemo seznam vseh instanc na projektu `admin`. Na voljo imamo naslednje začetne podatke:

- uporabniško ime: `admin`,
- geslo: `stack`,
- projekt: `admin` in
- URL strežnika za identiteto: `http://192.168.0.102:5000`.

Za dostop do vmesnika API bomo uporabili orodje CURL [8], ki ga lahko uporabljamo za prenos podatkov preko različnih protokolov v ukazni vrstici.

Najprej se moramo prijaviti v strežnik za identiteto, da pridobimo žeton za nadaljnjo uporabo storitev.

```
curl -d  '{"auth":{"passwordCredentials":{"username": "admin",
  "password": "stack"}}}' -H "Content-type: application/json"
  http://192.168.0.102:5000/v2.0/tokens
```

Nazaj dobimo naslednji odgovor, v katerem sta med drugim vidna žeton in čas njegove veljavnosti:


```
{
    "access": {
        "metadata": {
            "is_admin": 0,
            "roles": []
        },
        "serviceCatalog": [],
        "token": {
            "expires": "2014-01-24T19:06:31Z",
            "id": "MIIC8QYJKoZIhvcNAQcCoIIC4jCCAt4CAQExCTAH
                   BgUrDgMCGjCCAUcGCSqGSIb3DQEHAaCCATgEggE0
                   eyJhY2Nlc3MiOiB7InRva2VuIjogeyJpc3N1ZWRf
                   OiAiMjAxNC0wMS0yM1QxOTowNjozMS4xMjEyNjQi
                   CAiZXhwaXJlcyI6ICIyMDE0LTAxLTI0VDE5OjA2O
                   ...
                   AQOjE8mHYFNH+46BI5LdMhD9EFTTDWM1Y3KAjQ==",
            "issued_at": "2014-01-23T19:06:31.121264"
        },
        "user": {
            "id": "a71ab5e4286040e7a5b95001c2a359e3",
            "name": "admin",
            "roles": [],
            "roles_links": [],
            "username": "admin"
```

```
        }
    }
}
```

S tem žetonom nato izpišemo seznam projektov, do katerih imamo dostop, da dobimo njihove identifikacijske številke. Namesto spremenljivke TOKEN v spodnjem ukazu vpišemo žeton iz rezultata prejšnjega ukaza:

```
curl -H "X-Auth-Token:TOKEN"
  http://192.168.0.102:5000/v2.0/tenants
```

Iz rezultata zgornjega ukaza dobimo identifikacijsko številko projekta `admin`, ki je b21cd5a65b9348a4bd9634a847f9fbe9. V ta projekt se nato ponovno prijavimo, da dobimo nov žeton za delo zgolj na tem projektu:

```
curl -k -X 'POST' -v http://192.168.0.102:5000/v2.0/tokens
  -d '{"auth":{"passwordCredentials":{"username": "admin",
  "password":"stack"},
  "tenantId":"b21cd5a65b9348a4bd9634a847f9fbe9"}}'
  -H 'Content-type: application/json'
```

Kot rezultat dobimo nov žeton in vse priklopne naslove (URL) za različne programske vmesnike (API) projekta. Med drugim dobimo podatek, da je URL za računski servis:

http://192.168.0.102:8774/v2/b21cd5a65b9348a4bd9634a847f9fbe9.

Na koncu uporabimo ukaz za izpis vseh instanc na projektu `admin`. Namesto spremenljivke TENANTTOKEN vpišemo žeton, ki smo ga dobili v prejšnjem ukazu:

```
curl -H "X-Auth-Token:TENANTTOKEN"
  http://192.168.0.102:8774/v2/
  b21cd5a65b9348a4bd9634a847f9fbe9/servers/detail
```

Kot rezultat dobimo seznam instanc in njihove osnovne podatke v JSON-obliki. V spodnjem primeru je izpisana samo prva instanca:


```
"servers": [
{
"OS-DCF:diskConfig": "MANUAL",
"OS-EXT-AZ:availability_zone": "nova",
"OS-EXT-SRV-ATTR:host": "ubuntu",
"OS-EXT-SRV-ATTR:hypervisor_hostname": "ubuntu",
"OS-EXT-SRV-ATTR:instance_name": "instance-00000003",
"OS-EXT-STS:power_state": 1,
"OS-EXT-STS:task_state": null,
"OS-EXT-STS:vm_state": "active",
"OS-SRV-USG:launched_at": "2014-01-20T15:49:42.000000",
"OS-SRV-USG:terminated_at": null,
"accessIPv4": "",
"accessIPv6": "",
"addresses": {
 "private": [
  {
   "OS-EXT-IPS-MAC:mac_addr": "fa:16:3e:77:64:84",
   "OS-EXT-IPS:type": "fixed",
   "addr": "10.0.0.4",
   "version": 4
  }
 ]
},
"config_drive": "",
"created": "2014-01-20T15:49:39Z",
"flavor": {
 "id": "42",
 "links": [
```

```
  {
   "href": "http://192.168.0.102:8774/
    b21cd5a65b9348a4bd9634a847f9fbe9/flavors/42",
   "rel": "bookmark"
  }
 ]
},
"hostId": "7508c8bd3a4ca8304c22b30dcdb
           1b35b65c1cf134868b840ef53ddef",
"id": "f6de8a3a-4b68-430e-89e9-abdd98582163",
"image": {
 "id": "83c2927d-e003-4c17-b7d6-63cf0d60c0b7",
 "links": [
  {
   "href": "http://192.168.0.102:8774/
            b21cd5a65b9348a4bd9634a847f9fbe9/
            images/83c2927d-e003-4c17-b7d6-63cf0d60c0b7",
   "rel": "bookmark"
  }
 ]
},
"key_name": null,
"links": [
 {
  "href": "http://192.168.0.102:8774/v2/
           b21cd5a65b9348a4bd9634a847f9fbe9/servers/
           f6de8a3a-4b68-430e-89e9-abdd98582163",
  "rel": "self"
 },
 {
  "href": "http://192.168.0.102:8774/
```

```
            b21cd5a65b9348a4bd9634a847f9fbe9/servers/
            f6de8a3a-4b68-430e-89e9-abdd98582163",
  "rel": "bookmark"
 }
],
"metadata": {},
"name": "web-server-01",
"os-extended-volumes:volumes_attached": [],
"progress": 0,
"security_groups": [
 {
  "name": "default"
 }
],
"status": "ACTIVE",
"tenant_id": "b21cd5a65b9348a4bd9634a847f9fbe9",
"updated": "2014-01-20T18:45:36Z",
"user_id": "a71ab5e4286040e7a5b95001c2a359e3"
},
...
```

# Poglavje 4

# Izdelava mobilne administratorske aplikacije

Za namen upravljanja oblaka OpenStack smo za mobilni operacijski sistem Android izdelali mobilno aplikacijo OpenStackMobile. Aplikacija za povezavo z oblakom uporablja OpenStack REST API, medtem ko obvestila iz oblaka sprejema preko sistema Google Cloud Messasing.

## 4.1 Izdelava aplikacij na platformi Android

Android je odprtokodni operacijski sistem za mobilne naprave, predvsem mobilne telefone in tablice z zasloni na dotik. Operacijski sistem razvija podjetje Google, ob vsaki novi različici pa je na voljo tudi njena izvorna koda. Gre za trenutno (2014) najbolj razširjen mobilni operacijski sistem, zato je bil tudi uporabljen za izdelavo mobilne aplikacije za upravljanje oblaka OpenStack.

Sistem temelji na jedru Linux, aplikacije zanj pa so napisane v Javi in tečejo v izvajalnem okolju Dalvik, ki je posebna različica javanskega izvajalnega okolja za mobilne naprave. Za razvoj aplikacij potrebujemo Android SDK, ki je prosto dostopen na spletu. Android SDK vsebuje razvojno okolje Eclipse in dodatna orodja, namenjena razvoju mobilnih aplikacij na platformi

Andorid (npr. emulator mobilnih naprav).

### 4.1.1 Osnovne lastnosti aplikacij v sistemu Android

Aplikacije za sistem Android so napisane v Javi in se prevedejo v izvajalno kodo s pomočjo razvojnega okolja Android SDK [2]. Rezultat prevoda je datoteka s končnico `.apl` (Android package), ki vsebuje vso prevedeno kodo aplikacije, vključno s podatki.

Ko paket namestimo v sistem, aplikacija živi v svojem peskovniku. Privzeto vsaka aplikacije teče pod svojim uporabniškim imenom v sistemu Linux in v svojem izvajalnem okolju, izolirana od ostalih aplikacij. Tako je poskrbljeno za večjo varnost, saj ima aplikacija dostop samo do tistih virov, ki jih potrebuje. Če želimo dostop do dodatnih virov, moramo to definirati v datoteki `AndroidManifest.xml`, kjer definiramo vse pravice, ki jih aplikacija potrebuje. Za mobilno aplikacijo bomo tako morali definirati vsaj pravico dostopa do interneta oziroma omrežja, da bomo lahko dostopali do vmesnika oblaka OpenStack.

### 4.1.2 Osnovni gradniki aplikacij

Aplikacije v sistemu Android uporabljajo naslednje osnovne gradnike [7]:

- Aktivnost (activity) - Aktivnost je ena stran uporabniškega vmesnika. Primer aktivnosti v naši aplikaciji je stran za prijavo v sistem ali stran s seznamom instanc na oblaku OpenStack.

- Servis (service) - Servis je komponenta, ki izvaja daljši proces v ozadju, ki ni primeren za izvajanje na uporabniškem vmesniku. Primer tega je servis, ki v ozadju bere podatke iz omrežja za kasnejši prikaz v aplikaciji.

- Ponudnik vsebine (content provider) - S ponudniki vsebine lahko dostopamo do različnih virov podatkov oziroma shranjujemo podatke,

denimo na podatkovno bazo SQLite ali na datotečni sistem. Ponudniki vsebine omogočajo tudi izmenjavo podatkov med aplikacijami.

- Sprejemnik (broadcast receiver) - Sprejemnik je komponenta, ki sprejema signale operacijskega sistema. Na ta način se lahko odzovemo na sistemske dogodke, kot so prižiganje in ugašanje zaslona, nizko stanje baterije in podobni. V naši aplikaciji je bil napisan sprejemnik za sprejemanje sporočil iz sistema Google Cloud Messasing.

### 4.1.3 `AndroidManifest.xml`

Datoteka `AndroidManifest.xml` je obvezni del vsake aplikacije in vsebuje njene osnovne nastavitve. V njej med drugim definiramo naslednje lastnosti:

- vse pravice, ki jih aplikacija potrebuje,
- vse osnovne gradnike, ki sestavljajo aplikacijo,
- najnižjo potrebno verzijo operacijskega sistema za delovanje aplikacije in
- strojne lastnosti, ki jih naprava potrebuje (npr. kamera, bluetooth, GPS itd.).

## 4.2 OpenStack API v mobilni aplikaciji

Do oblaka OpenStack iz mobilne aplikacije dostopamo z uporabo vmesnika OpenStack REST, do katerega lahko dostopamo z uporabo HTTP-metod, npr. GET. Da uporabljamo vmesnik REST, ne potrebujemo nobenih dodatnih knjižnic, temveč v osnovi zadostuje razred `HttpURLConnection`, ki je že del Jave. Ker pa je taka uporaba HTTP-metod precej primitivna in zapletena, je bila uporabljena javanska knjižnica Restlet [10], ki omogoča izdelavo tako strežnikov kot odjemalcev, ki temeljijo na tehnologiji REST. Knjižnica je odprtokodna in prosto dostopna na spletu. Na voljo so različne

edicije, med drugim tudi različica za operacijski sistem Android, ki je bila uporabljena v naši aplikaciji.

### 4.2.1 Primer uporabe programske knjižnice Restlet

Kot primer uporabe knjižnice Restlet bomo pokazali kodo, ki vrne podatke za instanco, ki teče na oblaku OpenStack. Na voljo imamo naslednje vhodne parametre:

- avtentikacijski žeton - predhodno smo se že avtenticirali,
- URL-naslov računskega strežnika API in
- ID instance, katere podatke želimo prebrati.

Prva stvar, ki jo naredimo, je nov objekt razreda `ClientResource`, ki bo naš odjemalec za HTTP-metode. Kot parameter mu podamo URL metode, ki jih predhodno sestavimo tako, da URL-naslovu računskega strežnika dodamo ID instance:

```
String lsUrl = asUrl + "/servers/" + asInstanceId;
ClientResource loClientResource = new ClientResource(lsUrl);
```

Nadalje je treba v glavo klica metode dodati avtentikacijski žeton, ki ga imamo shranjenega v spremenljivki `asToken`:

```
Series<Header> headers = (Series<Header>)
  loClientResource.getRequestAttributes()
  .get("org.restlet.http.headers");
if (headers == null) {
  headers = new Series<Header>(Header.class);
  loClientResource.getRequest().getAttributes()
  .put("org.restlet.http.headers", headers);
}
headers.add("X-Auth-Token", asToken);
```

Nato izvedemo metodo HTTP GET. Kot parameter ji podamo konstanto `MediaType.APPLICATION_JSON`, ki pove, da želimo podatke nazaj dobiti v JSON-obliki:

```
Representation loResult =
  loClientResource.get(MediaType.APPLICATION_JSON);

String lsResult = "";
try{
   lsResult = loResult.getText();
}catch(IOException e){
  e.printStackTrace();
}
```

Kot rezultat dobimo serializiran JSON-objekt, ki ga nato pretvorimo v pravi JSON-objekt in iz njega preberemo podatke instance:

```
String lsName = "";
String lsState = "";
String lsCreated = "";
String lsId = "";
String lsFlavorId = "";
JSONObject jsonObj = null;
JSONArray loServers = null;
JSONObject loServer = null;
JSONObject loFlavor = null;
try{
 jsonObj = new JSONObject(lsResult);
 loServer = jsonObj.getJSONObject("server");
 lsName = loServer.getString("name");
 lsState =  loServer.getString("status");
 lsCreated =  loServer.getString("created");
 lsId =  loServer.getString("id");
```

```
 loFlavor = loServer.getJSONObject("flavor");
 lsFlavorId = loFlavor.getString("id");
}catch(JSONException e){
  Log.e("OM","Napaka pri branju podatkov instance.",e);
}
```

## 4.3 Izdelava mobilne aplikacije

### 4.3.1 Osnovne lastnosti aplikacije

Aplikacija zahteva vsaj verzijo 4.1 operacijskega sistema Android. Drugih posebnih omejitev nima. Seveda je za normalno uporabo potreben dostop do interneta oziroma omrežja, kjer se nahaja oblak OpenStack.

Prvo stran aplikacije predstavlja prijava v sistem, kjer vpišemo svoje uporabniško ime, geslo ter projekt, za katerega želimo pregledovati podatke. Po uspešni prijavi nam aplikacija omogoča pregled instanc, navideznih diskovnih pogonov in slik sistemov. Za vsako instanco je omogočen tudi podroben pogled, kjer vidimo več podatkov o instanci in imamo omogočene tudi operacije nad njo.

Za povezavo z oblakom OpenStack je bila uporabljena knjižnica Restlet, kot je opisano v poglavju 4.2.1.

### 4.3.2 Ciljna skupina

Aplikacija je namenjena končnim uporabnikom oblaka OpenStack. Z njo ima administrator najete infrastrukture v oblaku, pregled in nadzor nad svojimi navideznimi strežniki in ostalimi viri, poleg tega pa tudi dobiva obvestila o dogodkih na svojo mobilno napravo.

### 4.3.3 Specifikacija zahtev

Aplikacija ponuja naslednje funkcionalnosti:

- prijavo v oblak OpenStack z uporabniškim imenom in geslom, ki je dodeljeno uporabniku,
- pregled seznama instanc navideznih strežnikov in njihovega stanja,
- pregled seznama navideznih pogonov in seznama slik navideznih strežnikov,
- podroben pogled nad instanco navideznega strežnika in pregled njenih specifikacij,
- ugašanje in prižiganje instanc navideznih strežnikov,
- prejemanje in pregled obvestil o naslednjih dogodkih v oblaku: prižiganje in ugašanje instance, prehod v stanje pripravljenosti (mirovanja) in prehod nazaj v stanje delovanja; začasna prekinitev delovanja instance in prehod nazaj iz začasne prekinitve v stanje delovanja.

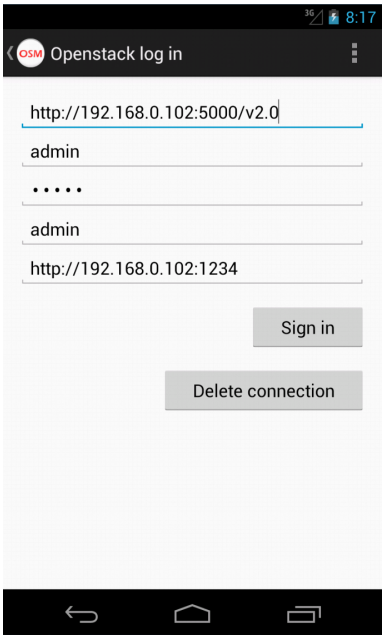
### 4.3.4 Pregled elementov aplikacije

#### Prijava v sistem

Ob odprtju aplikacije se najprej prikaže stran za prijavo v sistem. Na strani so naštete vse pretekle povezave, lahko pa tudi dodamo novo povezavo. Ob izbiri povezave ali odprtju nove se prikaže prijavna stran, kjer vpišemo naslednje podatke, kot prikazuje slika 4.1:

- URL Keystone servisa v oblaku OpenStack, na katerega se želimo povezati,
- uporabniško ime,
- geslo,
- projekt in
- URL servisa za registracijo sprejemanja sporočil.

Ob uspešni prijavi se podatki prijave vpišejo v zgodovino prijav kot nastavitev znotraj aplikacije v sistemu Android. Ob naslednji prijavi tako ni treba vseh podatkov vpisati ponovno, temveč samo izberemo obstoječo prijavo.



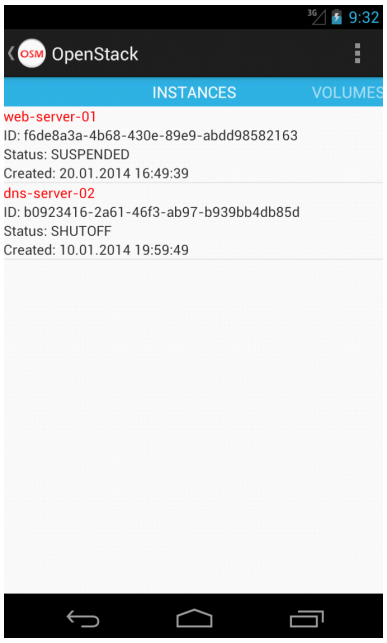

Slika 4.1: Prijava v sistem.

### Pregled projekta

Po prijavi v sistem se prikaže stran s tremi seznami za osnovni pregled nad projektom. Prikažejo se seznam instanc projekta, seznam navideznih diskovnih pogonov in seznam slik sistemov. Primer prikazuje slika 4.2.

Za pridobitev seznama instanc in ostalih podatkov je bil uporabljen OpenStack REST API, podobno kot je opisano v primeru v poglavju 3.2.3.

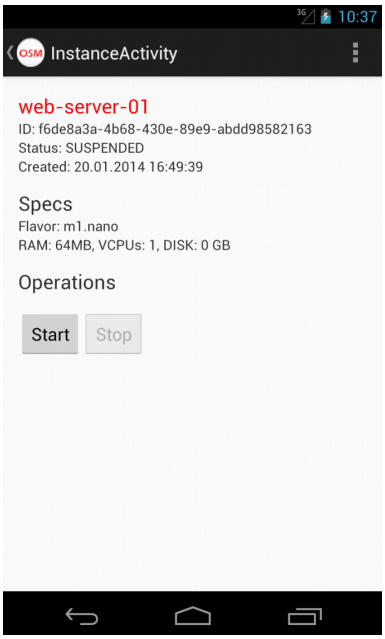
Če izberemo instanco, se odpre stran za podrobnejši ogled instance in operacije nad njo.




Slika 4.2: Seznam instanc na projektu.

### Pregled in operacije nad instanco

Stran za podrobnejši pregled instance vsebuje poleg njenih osnovnih podatkov tudi njene specifikacije ter operacije, ki jih lahko izvajamo nad njo. V naši aplikaciji je bilo implementirano ugašanje in prižiganje instance. Ustrezna operacija je omogočena samo, če je smiselna, torej že ugasnjene instance ni mogoče ponovno ugasniti. Primer strani za podroben pogled na instanco prikazuje slika 4.3.




Slika 4.3: Podroben pogled na instanco.

#### Sprejemanje sporočil

Aplikacija omogoča tudi sprejemanje sporočil iz oblaka OpenStack preko sistema Google Cloud Messaging. Več o tem v poglavju 4.4.

### 4.3.5 Izdelava strani za pregled projekta

Kot primer izdelave aktivnosti v aplikaciji, izdelani na platformi Android, bomo prikazali stran za pregled osnovnih podatkov projekta s seznamom instanc, navideznih diskovnih pogonov in slik sistemov.

Stran uporabniškega vmesnika v aplikacijah Android imenujemo aktivnost. Vsaka aktivnost je sestavljena iz dveh elementov:

- XML-datoteke, ki definira elemente grafičnega vmesnika in njihove lastnosti, ter
- datoteke z izvorno kodo, ki vsebuje dogodke in procedure, ki se prožijo znotraj aktivnosti.

Običajno zadostujeta ena XML-datoteka in javanski razred, izpeljan iz razreda `android.app.Activity`. V našem primeru pa gre za bolj kompleksno stran s tremi seznami, zato je bil uporabljen razred `FragmentActivity`, ki omogoča izdelavo strani uporabniškega vmesnika iz več sestavljenih elementov, ki jih imenujemo fragmenti.

Izdelanih je bilo pet XML-datotek za uporabniški vmesnik strani za pregled projekta:

- `activity_open_stack.xml` - Ogrodje strani in aktivnost, ki se prikaže na zaslonu po uspešni prijavi v aplikacijo. Vsebuje element `PagerTitleStrip`, ki omogoča razdelitev strani na jezičke, med kateremi se lahko premikamo.

- `openstack_list.xml` - Gradnik uporabniškega vmesnika za sezname. Vsebuje element `ListView`, ki se uporablja za prikazovanje seznamov na uporabniških vmesnikih. Uporabljen je bil na vseh treh seznamih (instance, slike, pogoni).

- `instance_item.xml` - Element uporabniškega vmesnika za eno instanco v seznamu. Vsebuje njene osnovne podatke.

- `volume_item.xml` - Element uporabniškega vmesnika za en navidezni diskovni pogon v seznamu. Vsebuje njegove osnovne podatke.

- `snapshot_item.xml` - Element uporabniškega vmesnika za eno sliko sistema v seznamu. Vsebuje njene osnovne podatke.

V ozadju uporabniškega vmesnika je bil izdelan razred `OpenStack`, ki je izpeljan iz razreda `FragmentActivity`. V metodi `onCreate` razreda se kreirajo instance razredov `InstanceFragment`, `VolumeFragment` in `SnapshotFragment`, ki prikazujejo sezname objektov iz oblaka OpenStack.

Spodnja koda prikazuje izsek kode razreda `InstanceFragment`. V metodi `onCreateView` preberemo seznam instanc v oblaku s klicem metode `OpenStackInstances.GetInstances`, ki nam vrne seznam instanc v spremenljivko tipa `InstanceData[]`. Metoda `OpenStackInstances.GetInstances`

pridobi podatke iz oblaka OpenStak preko REST-metod, kot je opisano v poglavju 4.2.1. Kot parameter dobi žeton za prijavo in URL komponente Nova v oblaku OpenStack, kar smo pridobili ob prijavi v sistem in imamo zapisano v globalni spremenljivki.

```
@Override
public View onCreateView(LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState) {
  View rootView;
  rootView = inflater.inflate(R.layout.openstack_list,
                              container, false);
  ListView rooViewList = (ListView)rootView;
  InstanceData[] loInstances
  = OpenStackInstances.GetInstances(isConnToken,
                                    isNovaURL);
  InstanceAdapter adapter
    = new InstanceAdapter(this, loInstances);
  rooViewList.setAdapter(adapter);
  ...
```

Nadalje kreiramo adapter tipa `InstanceAdapter`, v katerega pošljemo seznam instanc in ga priredimo seznamu, da vanj vpiše podatke instanc, ki jih želimo prikazati. V razredu `InstanceAdapter` definiramo metodo `getView`, ki nam generira en element seznama. Metoda v vse komponente elementa seznama, ki je definiran v datoteki `instance_item.xml`, vpiše podatke instance, ki jih hranimo v globalni spremenljivki `results` znotraj adapterja.

```
public View getView(int position,
                    View convertView,
                    ViewGroup parent) {
  View lView = convertView;
```

```
  if (lView == null) {
    lView = fInflater.inflate(R.layout.instance_item,
                              parent,
                              false);
  }
  TextView tvInstanceName
    = (TextView) lView.findViewById(R.id.InstanceName);
  TextView tvInstanceId
    = (TextView) lView.findViewById(R.id.InstanceID);
  TextView tvInstanceActive
    = (TextView) lView.findViewById(R.id.InstanceActive);

  tvInstanceName.setText(results[position].getInstanceName());
  tvInstanceId.setText("ID: "
    + results[position].getInstanceId());
  tvInstanceActive.setText("Status: "
    + results[position].getInstanceState());

  ...
```

Na koncu je treba glavno aktivnost definirati še v datoteki `AndroidManifest.xml`:

```
<activity
  android:name="com.andrejg.openstackmobile.OpenStack"
  android:label="@string/title_activity_open_stack"
  android:parentActivityName
    ="com.andrejg.openstackmobile.MainActivity" >
  <meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value
      ="com.andrejg.openstackmobile.MainActivity" />
```

```
</activity>
```

### 4.3.6 Uporabljene REST-metode

V aplikaciji so bile uporabljene naslednje REST-metode vmesnika API oblaka OpenStack.

Metode servisa za identiteto Keystone:

- POST-metoda `/v2/tokens` - Prijava v sistem. Kot rezultat dobimo žeton, ki ga uporabljamo za nadaljnjo avtentikacijo.
- GET-metoda `/v2/tenants` - Pridobimo seznam projektov, do katerih imamo dostop skupaj z URL-naslovi posameznih servisov.

Metode računskega servisa Nova:

- POST-metoda `/v2/(ID_projekta)/servers/(ID_strežnika)/action` - Izvedba akcije na navideznem strežniku (ustavitev, zagon).
- GET-metoda `/v2/(ID_projekta)/servers/(ID_strežnika)` - Pridobitev podrobnih podatkov o instanci navideznega strežnika.
- GET-metoda `/v2/(ID_projekta)/servers/detail` - Pridobitev podrobnih podatkov vseh instanc navideznih strežnikov.
- GET-metoda `/v2/flavors/(ID_tipa_instance)` - Pridobitev podrobnih podatkov o tipu instance.
- GET-metoda `/v2/images/detail` - Pridobitev seznama slik sistemov.

Metode servisa Cinder za shranjevanje podatkov na nivoju blokov:

- GET-metoda `/v2/(ID_projekta)/volumes/detail` - Pridobitev seznama navideznih diskovnih pogonov.

## 4.4 Google Cloud Messaging

Google Cloud Messaging (GCM) je spletni servis, ki razvijalcem mobilnih aplikacij na platformi Android omogoča izmenjavo podatkov med strežnikom in aplikacijo, nameščeno na napravi uporabnika. Običajno je to mobilni telefon ali tablični računalnik [11].

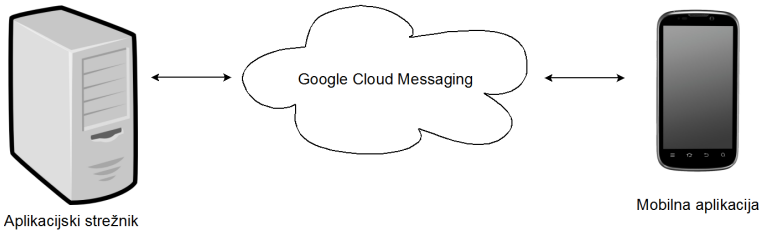
GCM sam poskrbi za vse vidike hranjenja sporočil v vrsti in dostavljanja sporočil končnim uporabnikom. Razvijalcu mobilne aplikacije ni treba skrbeti, ali je mobilna naprava prižgana in ali je v omrežju ter če aplikacija na mobilni napravi teče. GCM hrani sporočila v vrsti in jih dostavi v mobilno napravo, ko je ta dosegljiva. Ko dostavljeno sporočilo pride v napravo, lahko operacijski sistem zbudi ali zažene aplikacijo, ki ji je sporočilo namenjeno, tudi če ta ne teče. Novejše različice GCM-sistema omogočajo tudi pošiljanje podatkov iz naprave nazaj na strežnik. Servis je brezplačen in nima omejitev glede števila zahtevkov.

### 4.4.1 Arhitektura sistema GCM

Vsaka implementacija GCM-sistema je sestavljena iz treh komponent. To so:

1. mobilna aplikacija, ki zna sprejemati sporočila iz GCM-sistema,
2. aplikacijski strežnik, ki pošilja sporočila v GCM-sistem, preko katerega jih dostavlja v naprave, in
3. strežniki GCM, ki predstavljajo vmesni člen med našim strežnikom in mobilno aplikacijo na napravah naših uporabnikov.

Za prvo in drugo točko mora poskrbeti razvijalec sistema, za tretjo pa poskrbi Google s svojimi strežniki. Na strežnike GCM se lahko povezujemo preko HTTP- ali XMPP-protokola. Arhitekturo prikazuje slika 4.4.




Slika 4.4: Arhitektura sistema GCM.

### 4.4.2 Dopolnitev sistema OpenStack za pošiljanje obvestil

Na sistemu, kjer teče testna postavitev OpenStack, smo implementirali aplikacijski strežnik za pošiljanje sporočil iz sistema OpenStack v mobilno aplikacijo preko sistema GCM. Rešitev je bila implementirana v obliki dveh skript Python:

- skripte za registracijo naprav in
- skripte za pošiljanje sporočil na registrirane naprave preko sistema GCM.

Pri implementaciji so bila uporabljena naslednja orodja in programske knjižnice:

- programski jezik Python [19],
- knjižnica `Web.py` za izdelavo spletnih servisov REST v Pythonu [24],
- knjižnica Kombu, ki je uporabljena kot odjemalec za protokol AMQP v Pythonu [12], in
- knjižnica `python-gcm`, ki je uporabljena kot vmesnik za sistem Google Cloud Messasing v Pythonu [20].

### Registracija mobilnih naprav

Ko v mobilni aplikaciji registriramo napravo v sistem GCM, moramo aplikacijskemu strežniku poslati registracijski ID naprave, da bo imel informacijo, katerim napravam je treba pošiljati sporočila. V ta namen je bila izdelana skripta `gcm_server.py` v programskem jeziku Python [19].

Pri izdelavi skripte je bila uporabljena knjižnica `Web.py` [24], ki omogoča enostavno izdelavo spletnih servisov REST v jeziku Python. Izdelani sta bili dve POST-metodi:

- `[URL STREŽNIKA]\register` - Za registracijo naprave v sistem. Kot parameter `regId` dobi registracijski ID naprave.
- `[URL STREŽNIKA\unregister` - Za preklic registracije naprave v sistem. Kot parameter `regId` dobi registracijski ID naprave.

Spodnji izsek kode prikazuje implementacijo POST-metode za registracijo naprave, implementirano s pomočjo knjižnice `Web.py`. Program najprej prebere seznam naprav iz datoteke v spremenljivko `regFile`. V tem primeru registracijske ID zapisujemo v datoteko, v produkcijski implementaciji strežnika pa bi jih bilo bolj smiselno zapisovati v podatkovno bazo tipa SQL.

Potem preverimo, če v prebranem seznamu že imamo registracijski ID. Če ga ni, ga dodamo v seznam in zapišemo datoteko nazaj na disk. Registracijski ID je sedaj na voljo servisu za pošiljanje sporočil v sistem GCM.

Na podoben način je implementirana tudi metoda `unregister`, le da ta briše registracijske ID iz datoteke.

```
class register:
   def GET(self):
        return "Registracija z POST metodo na tem naslovu."
   def POST(self):
        data = web.input()
        # preberemo obstojece podatke
        if os.path.isfile(regFile):
```

```
                with open(regFile, 'rb') as f:
                        reg_list = pickle.load(f)
        else:
                reg_list = list()
        # dodamo ID naprave v seznam
        if data.regId not in reg_list:
                reg_list.append(data.regId)
                # Zapisemo datoteko
                with open(regFile, 'wb') as f:
                        pickle.dump(reg_list, f)
        return "OK"
```

### Pošiljanje sporočil v GCM

Drugi servis, ki je bil napisan, omogoča pošiljanje obvestil iz sistema OpenStack v GCM. Kot primer je bila napisana skripta za pošiljanje obvestil o stanju instanc v komponenti Nova. OpenStack uporablja AMQP-tehnologijo za prenos sporočil med različnimi komponentami sistema [14]. Kot posrednik, ki implementira tehnologijo AMQP, pa se uporablja bodisi RabbitMQ ali Qpid. V konkretnem primeru je bil uporabljen RabbitMQ.

Prva stvar, ki jo moramo narediti, je, da se naročimo na obvestila iz sistema. To naredimo tako, da v konfiguracijsko datoteko `nova.conf` komponente Nova dodamo spodnje vrstice:

```
notification_driver
 = nova.openstack.common.notifier.rpc_notifier
default_notification_level = INFO
notification_topics = notifications
notify_on_state_change = vm_and_task_state
notify_api_faults = true
```

Ko po spremembi nastavitev izvedemo ponovni zagon strežnika, smo pripravljeni na prejemanje obvestil. Obvestila lahko vidimo neposredno v

konzoli programa RabbitMQ [21], za pošiljanje sporočil v sistem GCM pa je bila napisana skripta v programskem jeziku Python s pomočjo knjižnice Kombu [12], ki jo lahko uporabimo kot vmesnik za branje sporočil poslanih preko AMQP-protokola.

Spodnji izsek kode prikazuje sprejemanje sporočil iz vrste, ki smo jo definirali v datoteki `nova.conf`. Na začetku inicializiramo vrsto, iz katere beremo:

```
nova_x = Exchange('nova', type='topic', durable=False)
info_q = Queue('notifications.info', exchange=nova_x,
                            durable=False,
               routing_key='notifications.info')
```

Nato vzpostavimo povezavo z vrsto preko AMQP-protokola. Ko sporočilo prispe, ga pošljemo naprej v metodo `process_msg`.

```
with Connection('amqp://guest:stack@localhost//') as conn:
    with conn.Consumer(info_q, callbacks=[process_msg]):
        while True:
            try:
                conn.drain_events()
            except KeyboardInterrupt:
                break
```

Nadalje implementiramo metodo `process_msg`, ki preveri, ali gre za tip sporočila, ki ga moramo poslati v sistem GCM. To določa seznam `gcmEvents`. Skripta nato kliče metodo `gcm_send`, ki oblikuje sporočilo in ga pošlje na Googlove strežnike:

```
gcmEvents = ['compute.instance.pause.end',
             'compute.instance.unpause.end',
             'compute.instance.power_off.end',
             'compute.instance.power_on.end',
             'compute.instance.suspend',
```

```
            'compute.instance.resume']


def process_msg(body, message):
    # preverimo tip eventa
    eventType = body['event_type']
    if eventType in gcmEvents:
                gcm_send(body)
    message.ack()
```

Na koncu implementiramo še metodo za pošiljanje sporočil v GCM-sistem. Za pošiljanje je bila uporabljena knjižnica `python-gcm` [20]. Na začetku je treba inicializirati GCM-vmesnik, tako da mu podamo ustrezen ključ Google API za strežniške aplikacije. V spodnjem primeru ni napisan dejanski ključ, ki je bil uporabljen. Ključ je mogoče dobiti brezplačno na spletni strani Google za razvijalce:

```
gcm = GCM("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")
```

Nadalje iz datoteke preberemo registracijske ID naprav, ki so se registrirale v naš sistem:

```
regFile = "device_ids.reg"
if os.path.isfile(regFile):
        with open(regFile, 'rb') as f:
                reg_ids = pickle.load(f)
```

Zadnji del postopka pa je klic metode `gcm_send`, ki oblikuje sporočilo iz podatkov, ki jih pošlje Nova v JSON-obliki in nato sporočilo posreduje mobilnim napravam preko GCM-sistema:

```
def gcm_send(body):
    eventType = body['event_type']
    project = body['_context_project_name']
```

```
computerName = body['payload']['display_name']

messageHead = project + ' - ' + computerName
messageBody = gcmEventsMessage[gcmEvents.index(eventType)]
 + ' (' + time.strftime("%d.%m.%y %H:%M:%S") + ')'

data = {'messageHead': messageHead,
              'messageBody': messageBody}
print 'broadcasting: ' +  messageHead + ',' + messageBody
response = gcm.json_request(registration_ids=reg_ids,
                            data=data)
```

### 4.4.3 Sprejemanje obvestil in prikaz v mobilni aplikaciji

Mobilna aplikacija OpenStackMobile je bila dopolnjena z možnostjo sprejemanja sporočil oblaka OpenStack preko sistema GCM. V aplikaciji je bilo treba narediti naslednje dopolnitve:

- dopolnitev datoteke `AndroidManifest.xml` z ustreznimi pravicami in definicijo servisa za sprejemanje sporočil,
- registracija naprave v sistem GCM,
- izdelava razreda za interakcijo z GCM in
- prikaz obvestil v sistemu.

#### Registracija naprave v sistem GCM

V sistem GCM se registriramo tako, da kličemo metodo register v razredu com.google.android.gcm.GCMRegistrar. Kot parameter ji pošljemo kontekst aplikacije ter spremenljivko SENDER_ID, v kateri imamo zapisan ID projekta v sistemu GCM, ki ga registriramo na Googlovi spletni strani za razvijalce.

```
GCMRegistrar.register(aoContext,SENDER_ID);
```

Po registraciji naprave v sistemu GCM bo ta vrnila registracijski ID naprave v naš razred GCMIntentService, ki ga moramo poslati nazaj na aplikacijski strežnik. To je opisano v naslednjem poglavju.

### Izdelava razreda za interakcijo z GCM

V aplikaciji je bil izdelan razred GCMIntentService, ki skrbi za interakcijo s sistemom GCM. V konstruktorju razreda registriramo ID projekta v sistemu GCM, ki je zapisan v spremenljivki SENDER_ID:

```
public class GCMIntentService extends GCMBaseIntentService {
    public GCMIntentService() {
        super(SENDER_ID);
    }
    ...
}
```

Nadalje je treba implementirati metodo onRegister, ki se kliče, ko se naprava uspešno registrira v sistem GCM. Kot parameter dobimo registracijski ID naprave, ki ga nato naprej pošljemo našemu aplikacijskemu strežniku. To se naredi z metodo ServerUtilities.register, ki ji pošljemo URL aplikacijskega strežnika in registracijski ID naprave. Metoda nato posreduje podatke aplikacijskemu strežniku s pomočjo ukaza HTTP POST.

```
@Override
protected void onRegistered(Context context,
                            String registrationId) {
  String lsUrl = ConnectionData.GetCurrentGCMUrl(context);
  if (lsUrl.length() > 0)
    ServerUtilities.register(context,
```

```
                                registrationId, lsUrl);
  }
```

Na podoben način je implementirana tudi metoda onUnregistered za odstranitev naprave iz sistema GCM.

Naslednja metoda, ki jo implementiramo, je onMessage, ki se proži, ko prispe sporočilo iz sistema GCM. V metodi lahko s funkcijo intent.getExtras() dobimo parametre sporočila, ki smo ga poslali v GCM iz aplikacijskega strežnika. Te parametre nato pošljemo naprej v metodo generateNotification, ki generira obvestilo v operacijskem sistemu Andorid.

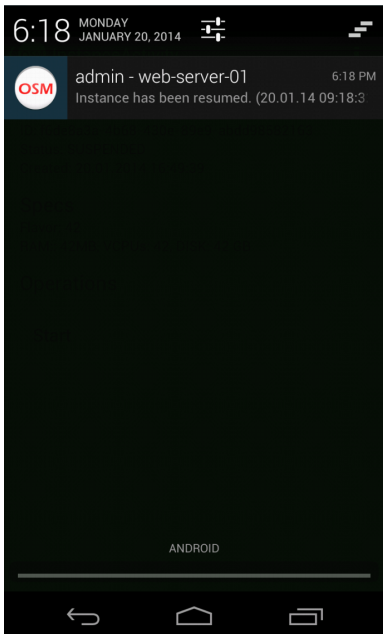
```
@Override
protected void onMessage(Context context, Intent intent) {
  String title = intent.getExtras().getString("messageHead");
  String message = intent.getExtras().getString("messageBody");
  generateNotification(context, title, message);
}
```

### Prikaz obvestil v sistemu

V razredu GCMIntentService je bila napisana še metoda generateNotification, ki generira obvestilo v operacijskem sistemu Android. Ob pritisku na obvestilo se zažene aplikacija in prikaže aktivnost MessagesActivity, ki je bila napisana za prikaz zgodovine obvestil, ki smo jih prejeli. Primer obvestila prikazuje slika 4.5.

```
private static void generateNotification(Context context,
                                         String title,
                                         String message) {
  int icon = R.drawable.ic_launcher;
  long when = System.currentTimeMillis();
  NotificationManager notificationManager =
    (NotificationManager)
```

```
    context.getSystemService(Context.NOTIFICATION_SERVICE);
  Notification notification = new Notification(icon,
                                                message, when);
  Intent notificationIntent = new Intent(context,
                                      MessagesActivity.class);
  notificationIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
    Intent.FLAG_ACTIVITY_SINGLE_TOP);
  PendingIntent intent =
    PendingIntent.getActivity(context, 0,
                                notificationIntent, 0);
  notification.setLatestEventInfo(context, title,
                                    message, intent);
  notification.flags |= Notification.FLAG_AUTO_CANCEL;
  notificationId = notificationId + 1;
  notificationManager.notify(notificationId, notification);
}
```




Slika 4.5: Obvestilo, prejeto iz OpenStack preko GCM. Instanca web-server-01 na projektu admin je bila prižgana.

## 4.5 Primerjava z ostalimi aplikacijami

### 4.5.1 Primerjava z ostalimi mobilnimi aplikacijami

Pred začetkom pisanja diplomske naloge je bila edina mobilna aplikacija za administracijo oblaka OpenStack Cloud Manager, podjetja 2kit consulting, ki pa ni specializirana za OpenStack, ampak ponuja dostop do vrste različnih oblakov, med drugim Amazon EC2, VMWare vSphere, Eucalyptus in drugih. Aplikacija omogoča pregled instanc navideznih strežnikov v oblaku in osnovne operacije nad njimi. Na voljo je za operacijska sistema Android in iOS.

V času pisanja diplomske naloge se je na trgu pojavila še aplikacija Stack Mobile, podjetja N3Infinity, za operacijski sistem Android, ki je namenjena administraciji oblaka OpenStack. Gre za trenutno najbolj dovršen mobilni

odjemalec za OpenStack. Omogoča naslednje funkcionalnosti:

- delo z več uporabniškimi računi in projekti,
- pregled instanc in operacije nad instancami navideznih strežnikov,
- pregled in upravljanje navideznih diskovnih pogonov,
- pregled porabe virov ter
- pregled in upravljanje navideznih omrežij.

V primerjavi z ostalimi mobilnimi aplikacijami je naša aplikacija OpenStackMobile, izdelana v okviru diplomske naloge, trenutno edina, ki ima implementiran sistem obveščanja o dogodkih v oblaku.

### 4.5.2 Primerjava z ostalimi aplikacijami za nadzor oblaka OpenStack

V primerjavi z ostalimi načini administracije oblaka je mobilna aplikacija precej omejena, saj njen nabor funkcionalnosti ni tako bogat kot pri nadzorni plošči Horizon (opisano v poglavju 3.2.2) ali administraciji preko ukazne vrstice (opisano v poglavju 3.2.1). To tudi ni njen namen, saj je mobilna aplikacija namenjena predvsem za hitra opravila, ko administrator nima drugih možnosti administracije, sistem za obveščanje pa poskrbi, da je vedno na tekočem o dogodkih na instancah navideznih strežnikov, za katere je zadolžen.

## 4.6 Vzpostavitev testnega okolja

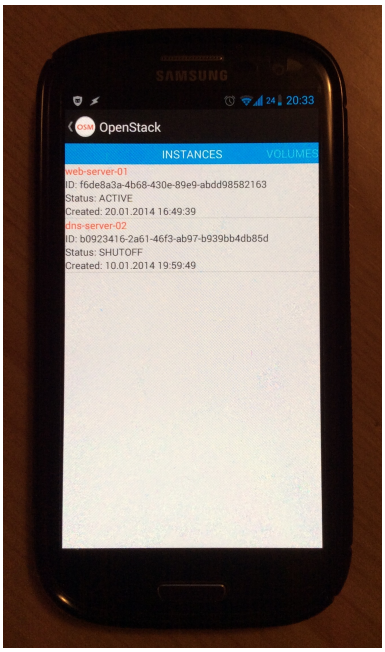
Testno okolje OpenStack je bilo postavljeno v okolju GNU/Linux, in sicer je bila uporabljena distribucija Ubuntu Linux Server 12.04 LTS [22]. Za testne namene je bila distribucija nameščena na virtualni strežnik v okolju VMWare na računalniku z operacijskim sistemom Windows 7. Virtualnemu strežniku sta bila dodeljena 2 procesorja, 4 GB RAM-a in 20 GB prostora na disku.

Za postavitev oblaka OpenStack je bila uporabljena skripta Devstack [6]. Skripta je napisana v jeziku Bash in omogoča enostavno postavitev razvojnega okolja OpenStack, ki lahko teče bodisi na enem ali več računalnikih. V konkretnem primeru je bil celoten oblak nameščen na enem računalniku. Devstack ni namenjen za produkcijska okolja, temveč za razvijalce, za testiranje novih funkcionalnosti in za demonstracije.

Za testiranje mobilne aplikacije je bil uporabljen emulator operacijskega sistema Android, ki ga dobimo poleg orodij za razvoj aplikacij Android - Android Developer Tools [5], ki so prosto dostopna na spletu. V emulatorju je bila kreirana virtualna mobilna naprava z naslednjimi lastnostmi:

- kot osnova je bila vzeta naprava Google Nexus 4,
- ločljivost zaslona: 768 X 1280,
- operacijski sistem: Android 4.1.2,
- pomnilnik: 512 MB.

Mobilna aplikacija je bila testirana tudi na fizični napravi, in sicer Samsung Galaxy S3, kot prikazuje slika 4.6.




Slika 4.6: Aplikacija OpenStackMobile na telefonu Samsung Galaxy S3.

# Poglavje 5

# Zaključek

Z nenehnim širjenjem računalništva v oblaku in s selitvijo vedno večjega števila produkcijskih sistemov v oblak narašča tudi potreba po čim bolj enostavni administraciji informacijskih virov v oblaku. Mobilne aplikacije so lahko eden izmed pripomočkov, ki zagotavljajo, da so administratorji čim prej obveščeni o dogajanju v oblaku in lahko v najkrajšem možnem času odpravijo morebitne težave.

Diplomsko delo predstavlja osnovo za izdelavo mobilnih aplikacij za administracijo oblačnih sistemov. Izdelana aplikacija ne omogoča vseh funkcionalnosti nadzorne plošče OpenStack, temveč je narejena kot primer, na katerem je mogoče zgraditi bolj celovito aplikacijo. Za bolj celovito aplikacijo bi bilo treba implementirati klice vseh funkcij programskega vmesnika oblaka OpenStack. Hkrati bi bilo treba tudi bolj tesno z oblakom OpenStack integrirati modul za pošiljanje sporočil v mobilne naprave, ki ga je trenutno treba zaganjati ločeno.

Izvorna koda aplikacije, ki je bila izdelana, je dostopna na spletnem naslovu: https://github.com/AndrejG82/openstackmobile.

# Seznam uporabljenih kratic

AMQP - Advanced Message Queuing Protocol
API - Application Programming Interface
AWS - Amazon Web Services
GCM - Google Cloud Messasing
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
IaaS - Infrastracture as a Service
ISP - Internet Service Provider
JSON - JavaScript Object Notation
LDAP - Lightweight Directory Access Protocol
PaaS - Platform as a Service
REST - Representational state transfer
RvO - Računalništvo v oblaku
SaaS - Software as a Service
SDK - Software Development Kit
SOAP - Simple Object Access Protocol
SSO - Single Sign On
URL - Uniform Resource Locator
VIM - Virtual Infrastructure Manager
VM - Virtual Machine
WSDL - Web Services Description Language
XML - Extensible Markup Language
XMPP - Extensible Messaging and Presence Protocol

# Literatura


[1] T. Erl, Cloud Computing: Concepts, Technology and Architecture, Prentice Hall, 2013 (stran 32-37, 58-61, 63-70, 73-78, 79-111, 213-227)

[2] R. Meier, Professional Android 4 Application Development, Wrox, 2012

[3] (2014) Amazon EC2. Dostopno na: https://aws.amazon.com/ec2/

[4] (2014) Amazon S3. Dostopno na: https://aws.amazon.com/s3/

[5] (2014) Android Developer Tools. Dostopno na: https://developer.android.com/tools/index.html

[6] (2014) Devstack - Configuration. Dostopno na: http://devstack.org/configuration.html

[7] (2014) Dokumentacija Android. Dostopno na: https://developer.android.com/guide/index.html

[8] (2014) Dokumentacija CURL. Dostopno na: http://curl.haxx.se/docs/

[9] (2014) Dokumentacija OpenStack. Dostopno na: http://docs.openstack.org/

[10] (2014) Dokumentacija programske knjižnice Restlet. Dostopno na: http://restlet.org/learn/guide/2.1/

[11] (2013) Google Cloud Messaging for Android. Dostopno na: https://developer.android.com/google/gcm/index.html

[12] (2013) Kombu knjižnica za sporočanje. Dostopno na: https://pypi.python.org/pypi/kombu

[13] (2014) Microsoft Private Cloud. Dostopno na: http://www.microsoft.com/en-us/server-cloud/solutions/virtualization-private-cloud.aspx

[14] (2014) Navodila za razvijalce OpenStack. Dostopno na: http://docs.openstack.org/developer/openstack-projects.html

[15] (2014) OpenStack. Dostopno na: http://www.openstack.org/

[16] (2014) OpenStack API. Dostopno na: http://api.openstack.org/

[17] (2014) OpenStack Horizon. Dostopno na: https://github.com/openstack/horizon

[18] (2011) OpenStack Overview Datasheet. Dostopno na: http://www.openstack.org/downloads/openstack-overview-datasheet.pdf

[19] (2013) Python dokumentacija. Dostopno na: http://www.python.org/doc/

[20] (2013) Python klient za Google Cloud Messaging. Dostopno na: https://github.com/geeknam/python-gcm

[21] (2013) RabbitMQ sistem za sporočanje. Dostopno na: http://www.rabbitmq.com/documentation.html

[22] (2012) Ubuntu 12.04 - Ubuntu Server Guide. Dostopno na: https://help.ubuntu.com/12.04/serverguide/index.html

[23] (2014) VMWare vCloud. Dostopno na: http://www.vmware.com/products/vcloud-suite/

[24] (2013) Web.py knjižnica. Dostopno na: http://webpy.org/

[25] (2014) Wikipedia, Cloud computing. Dostopno na: https://en.wikipedia.org/wiki/Cloud_computing

[26] (2014) Windows Azure. Dostopno na: http://www.windowsazure.com/