Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Boris Savić

Enoten programski vmesnik za dostop do različnih oblakov

diplomsko delo na univerzitetnem študiju

Mentor: doc. dr. Mojca Ciglarič

Ljubljana, 2013

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: 00068/2013
Datum: 02.04.2013

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

Kandidat: **BORIS SAVIĆ**

Naslov: **ENOTEN PROGRAMSKI VMESNIK ZA DOSTOP DO RAZLIČNIH OBLAKOV**

ACCESS TO DIFFERENT CLOUD PLATFORMS THROUGH UNIFORM INTERFACE

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Preučite različne oblačne platforme, ki nudijo storitve po modelu "infrastruktura kot storitev" - IaaS. Primerjajte programske vmesnike za dostop do platform OpenStack in VmWare vCloud. Nato poiščite in kritično ovrednotite nekaj knjižnic, ki skušajo programske vmesnike za dostop do teh platform poenotiti. Zasnujte lastno knjižnico za dostop do oblačnih platform IaaS, ki bo poleg upravljanja z računskimi enotami podpirala tudi pregledovanje in izbiro različnih nivojev storitve - ponudb SLA različnih ponudnikov. Knjižnica naj vključuje spletni vmesnik za preprosto upravljanje različnih oblačnih platform preko brskalnika. Rešitev ustrezno pretestirajte in v zaključku pojasnite njene prednosti in slabosti.

Mentor:
M. Ciglarič
doc. dr. Mojca Ciglarič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Boris Savić, z vpisno številko **63100311**, sem avtor diplomskega dela z naslovom:

Enoten programski vmesnik za dostop do različnih oblakov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič,
- 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 12. septembra 2013 Podpis avtorja:

Zahvalil bi se mentorici doc. dr. Mojci Cigralič za strokovno pomoč in podporo pri izdelavi diplomske naloge. Zahvala gre tudi staršema za podporo skozi vsa leta študija.

Staršem.

Kazalo

Povzetek

Tema diplomskega dela je računalništvo v oblaku, in sicer področje programskih vmesnikov različnih platform, ki omogočajo storitve tipa IaaS. V diplomskem delu bomo spoznali platformi OpenStack in VmWare vCloud ter primerjali njuna programska vmesnika. Spoznali bomo, da so programski vmesniki lahko precej različni, zato bomo v nadaljevanju pregledali nekaj možnih rešitev za premoščanje razlik med posameznimi platformami. Predstavljeni bosta programski knjižnici LibCloud ter jClouds. Ker omenjeni knjižnici omogočata le enostavno uporabo različnih ponudnikov gostovanja v oblaku, ne pa tudi izbire ponudb SLA, smo razvili svojo knjižnico XCloud. Knjižnica, poleg klasičnih funkcionalnosti za upravljanje z računskimi enotami, podpira tudi enostaven pregled in izbiro ponudb SLA različnih ponudnikov. V sklopu XClouda smo razvili tudi preprost spletni vmesnik, ki omogoča upravljanje različnih platform preko brskalnika.

Ključne besede

Računalništvo v oblaku, enoten programski vmesnik, SLA, računske enote, jClouds, LibCloud, OpenStack, vCloud

Abstract

This diploma thesis focuses on cloud computing, specifically on the area of IaaS oriented clouds. We introduce and compare API designs of two different cloud platforms - OpenStack and VmWare vCloud. Because cloud APIs can be significantly different, we take a deeper look at possible solutions for bridging the gap between different platforms. Two different solutions - LibCloud and jClouds are introduced. While both solutions overcome the differences in API designs, they don't provide sufficient support for selecting the right machine configurations based on provider SLAs. We present our own library - XCloud, for accessing different cloud platforms through a uniform interface that also offers its users the ability to perform SLA lookups and therefore select a cloud provider more easily. XCloud also provides a graphical interface accessible from within a web browser.

Keywords

Cloud computing, uniform API interface, SLA, compute instances, jClouds, LibCloud, OpenStack, vCloud

Poglavje 1

Uvod

Računalništvo v oblaku je v zadnjih nekaj letih v vzponu, saj omogoča ponujanje storitev na svetovnem spletu brez potrebe po lastni infrastrukturi in posledično tudi z manjšim finančnim proračunom. Računalniško infrastrukturo je možno najeti za potrebe uporabe različnih fizičnih ali pravnih oseb. Čeprav se nam danes to zdi samoumevno oziroma prav nič posebnega, pa je bil za nastanek računalništva v oblaku potreben velik miselni preskok. Tako, kot je možno uporabljati ali najeti različne storitve v vsakdanjem življenju, lahko dinamično najamemo in uporabljamo računalništvo v oblaku in s tem pridobimo določeno količino infrastrukture, ki jo v danem trenutku potrebujemo.

Z uporabo računalništva v oblaku se je pojavilo tudi nekaj težav in izzivov, slednji se nanašajo predvsem na množico programskih tehnologij in aplikacij, ki podjetjem in organizacijam omogočajo ponujanje storitev računalništva v oblaku. Različne platforme imajo prednosti in slabosti ter določene posebnosti, ki so lahko bolj ali manj primerne za podjetja ki omogočajo najem storitev, kot tudi za najemnike storitev.

Na tržišču je ogromno različnih ponudnikov računalniških storitev v oblaku, vendar to ne pomeni, da se mora razvijalec ali organizacija odločiti le za enega. Pogosto zakupi organizacija določeno količino storitev pri različnih ponudnikih. Razlogi za takšno odločitev so različni, najpogosteje pa želi

organizacija testirati zmogljivost najete infrastrukture oziroma storitve in primerjati različne ponudnike med seboj, preden se odloči za najem njihovih storitev v celoti. Drugi najpogostejši razlog je, da si želi organizacija zagotoviti nekakšno varnost, predvsem zaradi stabilnosti in zanesljivosti ponudnikove infrastrukture in storitev. Kljub temu, da so storitve računalništva v oblaku vedno dostopne, je bilo možno zabeležiti občasno nedelovanje infrastrukture (nedelovanje podatkovne baze, izpad strežnikov) tudi pri večjih ponudnikih storitev v oblaku.

Eden izmed izzivov je torej tudi, kako zagotoviti delovanje aplikacij oziroma programske opreme tako, da bo le ta nemoteno delovala pri več različnih ponudnikih (hkrati) ali pri zamenjavi ponudnika. Drugi izziv pa je sama izbira ponudnika, ta je trenutno prepuščena uporabniku. Pojavlja pa se potreba po bolj transparentnem pregledu ponudb in posledično lažji, morda tudi avtomatizirani izbiri ustreznega ponudnika.

Na tržišču trenutno prevladujeta dve knjižici - LibCloud ter jClouds, ki dokaj uspešno rešujeta prvi izziv, vendar pa ne nudita dovolj podpre pri transparentnem pregledu pogojev najema storitev v oblaku.

Cilj diplomskega dela je razviti enoten programski vmesnik za dostop do različnih oblakov. Razviti programski vmesnik bo vseboval tudi dodatne lastnosti, ki jih druge tovrstne rešitve nimajo, kot so preprosta izbira ponudnika na podlagi ponudb SLA ter spletni vmesnik, ki bo omogočal upravljanje različnih oblačnih platform preko brskalnika.

Poglavje 2

Računalništvo v oblaku

V zgodovini računalništva je bilo zaznati večji napredek vsakih pet ali deset let, zadnje desetletje pa je zaznamoval predvsem pojav računalništva v oblaku. Organizacijam in razvijalcem je omogočilo cenovno ugoden dostop do velikega števila računskih enot (ang. compute instances). Čeprav velja računalništvo v oblaku za relativno nov pojem, se zanj uporablja tehnologija, poznana že nekaj desetletij, in sicer virtualizacija. Računalništvo v oblaku je torej pravzaprav nov način poimenovanja skupne uporabe fizičnih virov (resursov) in je moderna alternativa postavljanju lastne infrastrukture.

Računalništvo v oblaku je nabor omrežnih storitev, ki omogočajo skalabilno, kvalitetno, pogosto personalizirano, cenovno ugodno rabo računskih enot, ki so preprosto dostopne [1].

Ponudba storitev v oblaku se je z leti razvijala in abstrahirala tako, da so se izoblikovali trije načini ponujanja storitev [2]:

1. IaaS – Infrastructure as a Service – ponujanje v najem virtualizirane infrastrukture in osnovne programske opreme (operacijski sistem),

2. PaaS – Platform as a Service – abstrakcija storitev IaaS, kjer se v najem ponujata tako strojna oprema kot določena količina programske opreme – npr. integracija programskih funkcij ali podatkovne baze,

3. SaaS – Software as a Service – najvišja abstrakcija, kjer organizacija

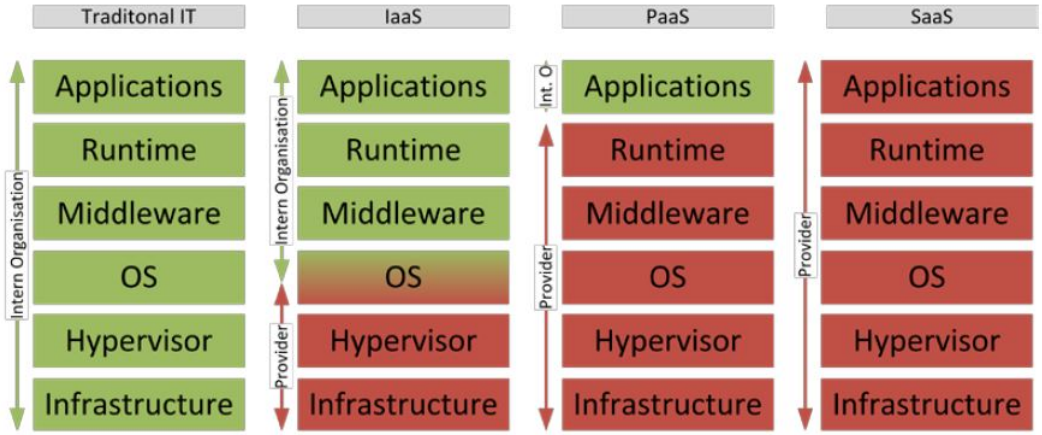
uporablja programsko opremo preko spletnega ali programskega vmesnika.

2.1 Prednosti

Računalništvo v oblaku prinaša mnogo prednosti tako s tehničnega kot tudi finančnega vidika organizacije, ki se odloči za najem storitev v oblaku. Nakup in postavitev lastne infrastrukture – lastnega podatkovnega centra, zahteva visoko začetno investicijo in veliko skrbnega načrtovanja, dve stvari, ki si jo mnoga podjetja na začetku svoje poti ne morejo privoščiti. Pri postavitvi lastne infrastrukture je potrebno zagotoviti dovolj računske moči, da bo podjetje lahko nemoteno ponujalo svoje storitve tudi ob občasnih večjih računskih zahtevah, kot tudi poskrbeti, da je podatkovni center ves čas primerno obremenjen. Z odločitvijo za najem računalniških storitev v oblaku postane problem načrtovanja lastne infrastrukture in predvidevanja njene obremenitosti povsem nerelevanten, saj lahko podjetje najame točno toliko strežnikov, kot jih za svoje nemoteno delovanje potrebuje. Organizacija ima namreč možnost dinamične alokacije potrebnih resursov, kar pomeni, da imajo v lasti oziroma v najemu vedno točno toliko računskih enot, kot jih potrebujejo.

Poleg precej zmanjšanih finančnih stroškov podjetja prinaša računalništvo v oblaku tudi mnoge druge prednosti [2]:

- Organizacija lahko hrani večje količine podatkov kot na lastni infrastrukturi,
- Sistemskim tehnikom ni treba več skrbeti za nedelujočo strojno opremo,
- Fleksibilnost pri izbiri ponudnika – podjetje lahko najame računske enote pri različnih ponudnikih,
- Hitrejša postavitev novih računskih enot – na voljo so vnaprej pripravljene sistemske slike s pred nameščeno potrebno programsko opremo,
- Elastično dodajanje ali odvzemanje resursov.



Slika 2.1: Primerjava različnih tipov storitev v oblaku. [3]

2.2 Primerjava različnih tipov storitev

Na sliki 2.1 lahko vidimo primerjavo vseh treh prevladujočih storitev v oblaku - IaaS, PaaS ter SaaS z lastno infrastrukturo. S slike je razvidno, da z vsakim nivojem abstrakcije ponujenih storitev skrbi organizacija oziroma razvijalec za manjši nabor komponent, kar je dobro, saj za dosegljivost in pravilno delovanje ostalih komponent skrbi in zagotavlja ponudnik storitve.

Opazimo lahko tudi, da mora organizacija pri lastni infrastrukturi skrbeti prav za vse komponente, medtem ko na najvišjem nivoju SaaS podjetje le najema programsko opremo, ki se izvaja v oblaku in tako skrbi le za reševanje svojega poslovnega problema ter se neposredno ne ukvarja s strojno ali programsko opremo.

2.3 Slabosti IaaS

Ker je cilj diplomske naloge predstaviti rešitev dostopa do programskih vmesnikov različnih ponudnikov IaaS storitev, si na kratko poglejmo nekaj še nekaj njenih slabosti. Med slabosti lahko v nekaterih primerih štejemo [3]:

- Dejanska lokacija podatkov ni vedno znana *(transparentna)* niti v jav-

nih niti zasebnih oblakih,

- zanašanje na dostopnost infrastrukture in omrežne povezave,
- v nekaterih primerih podatki različnih uporabnikov infrastrukture niso med seboj dovolj ločeni,
- v primeru slabe konfiguracije lahko pride do neavtoriziranega dostopa do podatkov drugih uporabnikov.

Problemi in razprave o slabostih IaaS, kot tudi računalništva v oblaku nasplošno, so predvsem pravne narave in se dotikajo predvsem pojmov o lastništvu podatkov, varnosti podatkov pred nedovoljenim dostopom tretjih oseb itd.

Poglavje 3

Platforma OpenStack

Platforma OpenStack je začela nastajati leta 2010 z željo, da bi organizacije lahko ponujale in uporabljale računalniške storitve v oblaku, ki bi tekle na klasični strojni opremi. To bi pomenilo, da za postavitev lastnega zasebnega oblaka organizacija ne bi potrebovala nadgradnje svoje obstoječe strojne opreme, saj bi bil poseg le programske narave.

Velika prednost platforme OpenStack je njena odprtokodnost, vendar je potrebno poudariti, da za razvojem platforme ne stoji samo skupina prostovoljcev, saj ima OpenStack široko podporo tudi v industriji. Pri razvoju sodeluje več kot 200 podjetij, na seznamu teh podjetij najdemo tudi številna priznana imena, kot sta NASA in RackSpace [4].

Pri razvoju platforme OpenStack so želeli razvijalci ohraniti kar se da visoko stopnjo kompatibilnosti z Amazonovo platformo AWS EC2. S tem so dosegli, da so razvijalci in organizacije mnogo lažje in hitreje preselile svoje obstoječe programske rešitve, sprva razvite za platformo AWS EC2, na lastno postavitev OpenStack. Organizacije so tako lahko lažje odšle h konkurenčnemu podjetju, ki svoje storitve ponuja na platformi OpenStack.

Ravno zaradi velikega prizadevanja pri doseganju kompatibilnosti z AWS EC2 in posledično podobnega programskega vmesnika lahko platformi OpenStack pripišemo visoko razširjenost in sprejetost tako med razvijalci kot podjetji, ki ponujajo svojo infrastrukturo v najem.

Platforma OpenStack je dandanes široko razširjena v industriji in jo uporabljajo mnoga majhna in velika podjetja za postavitev privatnega in javnega oblaka. Med bolj znana podjetja in organizacije, ki uporabljajo platformo OpenStack, spadajo tudi [5]:

1. RackSpace,
2. HP,
3. AT&T,
4. Red Hat,
5. IBM.

OpenStack je sestavljen iz več modularno sestavljenih komponent in omogoča upravljanje z računskimi enotami, objektno/bločno shrambo podatkov (ang. Object/Block Storage) in omrežnimi storitvami. Ker se bomo v nadaljevnaju ukvarjali predvsem z računskimi enotami, si poglejmo nekaj lastnosti pripadajočega modula.

3.1 OpenStack Nova

Nova je programski modul platforme OpenStack, ki omogoča upravljanje in nadzor računskih enot. Skrbi za inicializacijo računske enote in njeno nemoteno delovanje. Do enot lahko vodstvo organizacije dostopa preko spletnega vmesnika, razvijalcem pa je dostopen tudi programski vmesnik, s katerim lahko nadzirajo, kreirajo in odstranijo računske enote. Nova je razvita v programskem jeziku Python in je zgrajena tako, da ponuja izjemno horizontalno skalabilnost na standardni strojni opremi.

Ob namestitvi platforme OpenStack nam le ta že vnaprej pripravi nekaj osnovnih konfiguracij računskih enot – vsaka vsebuje drugačen nabor zmogljivosti, izbira pa je sledeča:

- Tiny – (1 CPU, RAM 512 MB, Disk 10 GB),

- Small – (1 CPU, RAM 2048 MB, Disk 20 GB),
- Medium – (2 CPU, RAM 4096 MB, Disk 40 GB),
- Large – (4 CPU, RAM 8192 MB, Disk 80 GB),
- Xlarge- (8 CPU, RAM 16384 MB, Disk 160 GB).

Administrator storitve v oblaku ima možnost dodati ali odstraniti konfiguracijo, zato ponujajo pri ponudniku storitev v oblaku Rackspace tudi druge nastavitve, ki vsebujejo med drugim večje količine delovnega spomina.

Ob kreiranju nove računske enote razvijalec izbere operacijski sistem, katerih seznam zopet vnaprej pripravi administrator OpenStack platforme, ter ustrezno nastavi konfiguracijo sistema, ki podani nalogi najbolje ustreza. Najpogosteje razvijalci izberejo operacijski sistem iz različnih distribucij Linuxa. Ker do računskih enot Linuxa dostopamo preko SSH povezave, nam OpenStack omogoča generiranje javnega in zasebnega ključa (RSA) ob kreiranju nove računske enote za njen dostop. Nabor javnih in zasebnih ključev pri dostopu do računske enote nam omogoča večjo varnost kot dostop zavarovan le z geslom.

3.2 API

Kot že omenjeno, OpenStack ponuja nabor programskih API-jev, preko katerih lahko razvijalci dostopajo in upravljajo z vsemi funkcionalnostim, ki jih OpenStack ponuja. Programski API-ji temeljijo na tehnologiji REST, kar pomeni, da so dostopni preko HTTP protokola z izvajanjem ustreznih GET, POST, PUT, DELETE operacij na ustrezne spletne naslove.

Obstaja tudi vrsta uradnih in neuradnih knjižnic v različnih programskih jezikih, ki razvijalcem olajšajo delo tako, da ponujajo enostavnejše programske klice iz ustreznega jezika. Primera takšnih knjižnic v programskem jeziku Java sta:

- jClouds,

- OpenStack Java SDK.

Ker bomo v nadaljevanju pri razvoju svoje knjižnice za dostop do različnih oblakov uporabili OpenStack Java SDK, si poglejmo, kako z njeno pomočjo inicializiramo novo računsko enoto.

```
1     /**
2      * Start new virtual machine - image recognized by id.
          Hardware settings are stored in flavor.
3      * New VM will be named by name parameter.
4      * @param imageId - id of the virtual machine image
5      * @param flavor - machine configuration (tiny, large...)
6      * @param name - server name
7      * @param keypair - keypair name to use
8      * @return server - created server
9      * @throws CreateResourceException
10     */
11   public Server launchVirtualMachine(String imageId, String
       flavor, String name, String keypair)
12       throws CreateResourceException {
13
14     try{
15       ComputeClient compute = client.getComputeClient();
16       NovaServerForCreate vm = new NovaServerForCreate();
17       vm.setName(name);
18       vm.setImageRef(imageId);
19       vm.setFlavorRef(flavor);
20       vm.setKeyName(keypair);
21       Server s = compute.createServer(vm);
22       return s;
23
24     }catch(Exception e) {
25       e.printStackTrace();
26       throw new CreateResourceException(e.getMessage());
27     }
28   }
```

Listing 3.1: Uporaba knjižnice OpenStack SDK

Čeprav je iz primera izvzeta inicializacija povezave z OpenStackom, je razvidno, da potrebuje razvijalec le nekaj vrstic za postavitev nove računske enote, saj SDK sam poskrbi za pravilno sosledje REST klicev.

Poglavje 4

VmWare

Podjetje VmWare je bilo ustanovljeno leta 1998 in ima za seboj dolgo zgodovino na področju strežniških rešitev in virtualizacije [6]. Ravno zaradi svoje dolgoletne prisotnosti na področju virtualizacije je povsem naravno, da so med svoje produkte uvrstili tudi programsko družino vCloud, ki strankam omogoča postavitev in uporabo storitev v oblaku.

Programska rešitev vCloud je pogosto uporabljena znotraj organizacij za postavitev zasebnih oblakov, vendar pa najdemo na trgu tudi podjetja, ki nudijo možnosti najema računskih enot s pomočjo platforme vCloud. Eno takšnih podjetij je tudi slovensko podjetje Arctur [7], ki svojim strankam ponuja storitve na tehnologiji vCloud.

Programska družina vCloud je sestavljena iz več komponent [8]:

1. vSphere – ponuja virtualizirano inftrastrukturo, ki temelji na abstrahirani fizični strojni opremi,
2. vCloud Director – vstopna točka za najemnike rešitev v oblaku, ponuja izoliran večnajemniški model in omogoča razširljivost v javni oblak,
3. vCloud Connector – omogoča razdelitev dela med javnim in privatnim oblakom,
4. vCloud Networking and Security – ponuja programsko rešitev omrežnih in varnostnih storitev ter njihovo integracijo.

Veliko podjetij že uporablja programske rešitve VmWare za virtualizacijo in imajo torej nekaj lastne infrastrukture, zato jim je rešitev vCloud precej privlačna, saj omogoča preprosto nadgradnjo obstoječih sistemov in povezavo z javnim oblakom [8].

4.1 Računske enote

Proces kreiranja nove računske enote, ki jih ponuja platforma vCloud, se od procesa pri platformi OpenStack in podobnih konkurenčnih rešitvah razlikuje v nekaj točkah. Za začetek ima razvijalec povsem proste roke pri izbiri konfiguracije, saj lahko neodvisno izbere potrebno količino delovnega pomnilnika, diska in število CPE jeder. V orodju vCloud Director ima razvijalec sicer že vnaprej določene kataloge in predloge operacijskih sistemov, ki imajo CPE, RAM in disk količine že določene, vendar jih lahko poljubno spreminja po lastnih preferencah in zahtevah, medtem ko na primer na platformi OpenStack razvijalec lahko izbira le med vnaprej določenimi konfiguracijami in nima možnosti spreminjanja konfiguracije.

4.2 API

VmWare vCloud ponuja razvijalcem uradno podprto Javansko knjižnico oziroma SDK - *Software Development Kit*, preko katerega lahko razvijalci programsko komunicirajo z vCloud komponentami in tako kreirajo ter nadzorujejo računske enote oziroma ostale komponente. Omeniti velja tudi dodatno posebnost programskih klicev za platformo vCloud, ki omejuje število trenutno izvajajočih se akcij na le eno – v nekem danem trenutku se lahko izvaja le ena operacija. To omejitev bomo morali upoštevati pri razvoju podpore platformi VmWare vCloud v naši knjižnici XCloud [13].

Kreiranje nove računske enote je na platformi vCloud v primerjavi z OpenStack precej bolj kompleksen proces, saj zahteva kar nekaj vmesnih korakov. Proces kreiranja nove računske enote oziroma krajši izsek procesa

si poglejmo na sledečem primeru.

```
1   /**
2    * Launches new vApp from template on VMware cloud
3    *
4    * @param vAppRef
5    * @param vdc
6    * @param network
7    * @return
8    * @throws ComputeOperationException
9    * @throws VCloudException
10   */
11  private boolean createNewVappFromTemplate(ReferenceType
        vAppTemplateRef, Vdc vdc, String network)
12      throws ComputeOperationException {
13
14    InstantiationParamsType instantiationParamsType = setNetwork
        (network, vdc);
15    InstantiateVAppTemplateParamsType instVappTemplParamsType =
16        new InstantiateVAppTemplateParamsType();
17
18    instVappTemplParamsType.setName(vAppName);
19    instVappTemplParamsType.setSource(vAppTemplateRef);
20    instVappTemplParamsType.setInstantiationParams(
        instantiationParamsType);
21
22    Vapp vapp = vdc.instantiateVappTemplate(
        instVappTemplParamsType);
23    List<Task> tasks = vapp.getTasks();
24    if (tasks.size() > 0) {
25      tasks.get(0).waitForTask(0); //wait untill VM is created
26    }
27
28    configureVMsIPAddressingMode(vapp.getReference(), vdc);
29    return true;
30  }
```

Listing 4.1: Uporaba uradne vCloud knjižnice

V zgornjem primeru smo izpustili proces, ki je potreben za konfiguracijo omrežne povezave nove računske enote, kot tudi proces, kako pridemo do nekaterih ključnih objektov, kot sta *vdc* in *vAppTemplateRef*. Namen zgornjega primera in primera 3.1 je prikazati, kako zelo so si programski klici platform lahko različni.

Poglavje 5

Enoten programski vmesnik za dostop do različnih oblakov

Kot smo lahko opazili na primerih platform OpenStack in VmWare vCloud, so razlike med posameznimi platformami lahko precejšnje in posledično so tudi programski vmesniki med seboj precej različni. Te razlike predstavljajo precejšen problem za razvijalce, ki morajo zagotoviti nemoteno delovanje programske opreme tudi v primeru menjave ponudnika storitev.

Čeprav smo predstavili le platformi OpenStack in VmWare,je potrebno omeniti, da je različnih platform na trgu mnogo več. Nekatere med njimi so odprto-kodne druge lastniške. Naštejmo le nekaj bolj znanih ponudnikov oziroma platform:

- OpenStack,
- VmWare,
- GoGrid,
- OpenNebula,
- CloudSigma,
- CloudStack.

Menjava ponudnika storitev v oblaku pa ni edini razlog, zaradi česar bi želeli poenotiti uporabo programskih vmesnikov. Pogost je tudi primer, omenjen v uvodu, da želimo poganjati našo aplikacijo sočasno pri več različnih ponudnikih zaradi zagotavljanja višje stopnje dostopnosti v primeru izpadov ali pa le za potrebe testiranja učinkovitosti različnih ponudnikov.

Razvijalcem v takšnih primerih ne preostane veliko možnosti. Lahko prilagodijo svojo aplikacijo za vsakega ponudnika oziroma vsako platformo posebej, vendar takšna rešitev zahteva veliko dodatnega dela in časa, saj se mora razvijalec ponovno lotiti osnov in dobro poznati platformo ter dokumentacijo programskih vmesnikov. Naloga razvijalca je tudi programiranje funkcij, ki skrbijo za REST klice, saj veliko platform nudi le uradni REST vmesnik. Za uporabo v specifičnih programskih jezikih mora razvijalec sam poskrbeti za inicializacijo HTTP klicev in njihovo pravilno sosledje. Druga, precej bolj privlačna možnost je, da programer uporabi uradne oziroma neuradne knjižnice v programskem jeziku, v katerem razvija, vendar se mora tudi v tem primeru naučiti uporabljati več različnih knjižnic hkrati. Tretja in tudi najbolj razvijalcu prijazna možnost je, da uporabi knjižnico, ki že sama poskrbi za pravilno inicializacijo in izvajanje klicev ne glede na plaftormo oziroma ponudnika, pri kateri najema storitve. Takšna knjižnica bi znala upravljati z različnimi platformami z enotnim programskim vmesnikom le na podlagi konfiguracije, ki pa je večinoma trivialne narave. Na tak način se mora razvijalec seznaniti le s programskim vmesnikom ene same knjižnice, da bi obvladal in znal upravljati z oblaki različnih ponudnikov.

Eno takšnih knjižnic smo razvili tudi v okviru te diplomske naloge, predstavljena pa bo v naslednjih poglavjih. Za začetek si poglejmo nekaj obstoječih in uveljavljenih rešitev za preprosto upravljanje z različnimi oblaki.

5.1 LibCloud

Apachejeva knjižnica LibCloud je bila sprva razvita v okviru podjetja CloudKick [9], vendar je dokaj hitro prešla med neodvisne projekte, z Apachevo

Licenco (2.0). LibCloud je spisan v programskem jeziku Python in razvijalcem omogoča uporabno enotnega programskega vmesnika za dostop do različnih ponudnikov storitev v oblaku.

Knjižnica LibCloud nam omogoča kreiranje in upravljanje z računskimi enotami, poleg tega pa lahko s pomočjo LibClouda poganjamo tudi programske skripte, s katerimi poskrbimo za osnovno konfiguracijo programske opreme računske enote ter namestimo in poženemo lastno programsko opremo.

Poleg upravljanja z IaaS storitvami različnih ponudnikov so razvijalcu na voljo programski API-ji, ki mu omogočijo kreiranje in nadzor nad shrambo podatkov v oblaku (ang. Cloud Storage), koordinatorjem dela (ang. Loadbalancer as a Service, LbaaS) in DNSaaS (DNS as a Service).

5.1.1 Podprte platforme

Knjižnica LibCloud podpira nekaj več kot 20 različnih ponudnikov, oziroma različnih platform, vendar pa ne nudi podpore vsem funkcionalnostim pri vseh ponudnikih. Ker je z enotnim programskim vmesnikom praktično nemogoče pokriti vse funkcionalnosti, ki jih ponujajo različne platforme, so morali razvijalci knjižnice izbrati podmnožico funkcionalnosti, ki pokrije večino primerov uporabe in hkrati pokrije kar se da največje število različnih platform.

V okviru računskih enot so se razvijalci knjižnice LibCloud odločili podpreti sledeče funkcionalnosti [10]:

- list – vrne seznam vseh trenutno aktivnih instanc,
- reboot – uporabniku omogoči vnovični zagon določene instance,
- create – uporabniku omogoči kreiranje nove instance,
- destroy – uporabniku omogoči odstranjevanje obstoječe instance,
- images – vrne seznam vseh sistemskih slik, ki so na voljo,

- sizes – vrne seznam konfiguracij, ki so na voljo (količina pomnilnika, diska, CPE jeder, ...),
- deploy – uporabniku omogoči poganjanje skript za konfiguracijo sistema, potem ko je bil kreiran.

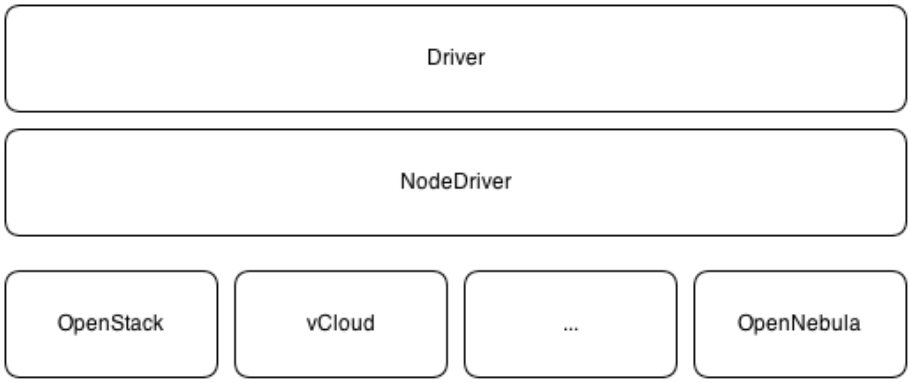
Ta podmnožica funkcionalnost je v veliki meri skupna večini knjižnicam za enoten dostop do različnih oblakov.
Na seznamu podprtih ponudnikov najdemo več kot 20 le teh, nekateri med njimi so:

1. OpenStack,
2. Amazon EC2,
3. GoGrid,
4. VmWare vCloud,
5. OpenNebula,
6. CloudStack.

5.1.2 Arhitektura

LibCloud ima relativno preprosto arhitekturo. Programska koda je razdeljena v 4 osnovne pakete, vsak od teh paketov pa skrbi za izvajanje ene od IaaS, DNSaaS, ali ostalih funkcionalnosti. Notranje arhitekture vsakega posameznega paketa so si med seboj zelo podobne. Na sliki 5.1 lahko vidimo, kako je v grobem strukturirana arhitektura za dostop do računskih enot.

Razvijalec izvaja vso komunikacijo prek vrhnje plasti – gonilnika (ang. driver), prek katerega vzpostavi povezavo z želenim ponudnikom. Vsi nadaljnji klici se nato posredujejo navzdol po strukturi od gonilnika do abstraktne implementacije *NodeDriver* in vse do dejanske implementacije izbranega ponudnika, ki je razširitev osnovnega razreda *NodeDriver*.



Slika 5.1: Groba razdelitev LibCloud arhitekture.

5.1.3 Primer uporabe

Poglejmo si še primer 5.1, ki prikazuje, kako s pomočjo knjižnice LibCloud vzpostavimo povezavo s ponudnikom Rackspace in kakšno je sosledje potrebnih ukazov za zagon nove računske enote.

```
1  from libcloud.compute.types import Provider
2  from libcloud.compute.providers import get_driver
3
4  RACKSPACE_USER = 'your username'
5  RACKSPACE_KEY = 'your key'
6
7  Driver = get_driver(Provider.RACKSPACE)
8  conn = Driver(RACKSPACE_USER, RACKSPACE_KEY)
9
10 # retrieve available images and sizes
11 images = conn.list_images()
12 # [<NodeImage: id=3, name=Gentoo 2008.0, driver=Rackspace  ...>,
       ...]
13 sizes = conn.list_sizes()
14 # [<NodeSize: id=1, name=256 server, ram=256 ... driver=
      Rackspace ...>, ...]
15
16 # create node with first image and first size
17 node = conn.create_node(name='test', image=images[0], size=sizes
      [0])
18 # <Node: uuid=..., name=test, state=3, public_ip=['1.1.1.1'],
      provider=Rackspace ...>
```

Listing 5.1: Uporaba knjižnice LibCloud [11]

Tu velja dodati, da povsem enako programsko kodo lahko uporabimo tudi za kreiranje računske enote, pri katerem drugem podprtem ponudniku. Spremeniti bi morali le začetne konfiguracijske vrstice, koda za izvajanje klicev za kreiranje računske enote pa bi ostala povsem enaka.

5.2 jClouds

jClouds je še ena izmed knjižnic, ki razvijalcu omogoča enostaven dostop do različnih oblakov prek enotnega programskega vmesnika. Vendar pa za razliko od LibCloud-a temelji na platformi JVM – *Java Virtual Machine* in torej omogoča upravljanje s storitvami v oblaku tistim razvijalcem, ki so za

razvojno okolje izbrali enega izmed jezikov, ki tečejo na JVM platformi – Java, Clojure, Scala itd.

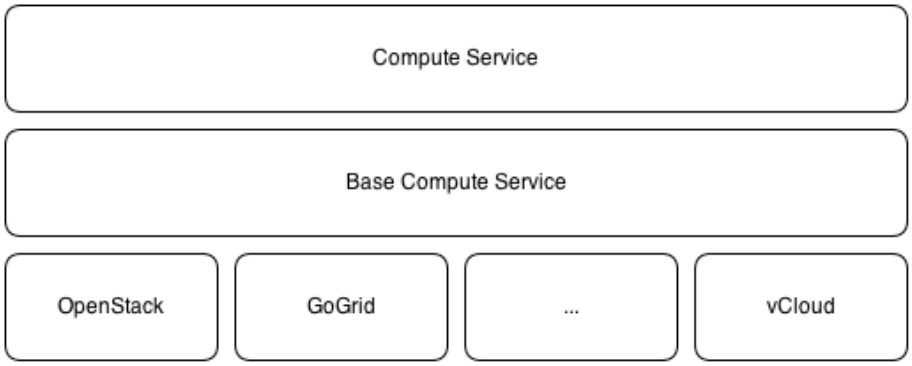
jClouds ponuja razvijalcu napredno upravljanje z različnimi ponudniki storitev v oblaku preko abstraktnega vmesnika, ki ponuja enoten programski vmesnik neodvisno od izbranega ponudnika. Poleg tega pa omogoča tudi upravljanje preko specifičnega programskega vmesnika, ki razvijalcu omogoči dostop do naprednih funkcionalnosti izbranega ponudnika. Prednost takšne zasnove je predvsem v tem, da lahko za večino skupnih operacij razvijalec uporablja abstraktne metode, skupne vsem ponudnikom. Če potrebuje dodatne specifične funkcionalnosti, pa jih lahko najde v specifičnem gonilniku v knjižnici jClouds in mu tako ni potrebno razviti tega dela funkcionalnosti.

Podobno kot knjižnica LibCloud tudi jClouds ponuja podporo različnim tipom storitev, med katerimi sta najbolj razviti in zreli podpori za računske enote in shramo podatkov v oblaku.

5.2.1 Podprte platforme

jClouds ima bogato in dobro razvito podporo za več kot 30 različnih ponudnikov storitev v oblaku, med katerimi so najbolj prepoznavni [12]:

- Amazon,
- Azure,
- GoGrid,
- NineFold,
- OpenStack,
- Rackspace,
- vCloud.



Slika 5.2: Groba razdelitev jClouds arhitekture.

5.2.2 Arhitektura

Dejanska realiziacija knjižnice jClouds je za odtenek bolj kompleksna v primerjavi s knjižnico LibCloud, ki smo si jo pogledali v prejšnjem poglavju. Vendar je v osnovi zgrajena iz podobnih komponent in tudi arhitekture obeh knjižnic sta si v grobem zelo podobni, kar je razvidno tudi iz slike 5.2.

Razvijalec vse klice izvaja preko osnovnega razreda *ComputeService*, ki ponuja abstraktne metode, skupne vsem ponudnikom. Klici se nato posredujejo v nižje plasti do potrebnih gonilnikov. V razredu *BaseComputeService*, iz katerega dedujejo vsi gonilniki posameznih platform, najdemo implementacije metod, ki so skupne vsem ponudnikom, medtem ko za bolj specifične operacije poskrbijo ustrezni razredi pripadajočih ponudnikov.

Kadar želi razvijalec uporabiti napredne funkcije, specifične glede na ponudnika, začne svoje klice izvajati nad razredom primernega ponudnika, kar pomeni, da direktno komunicira z najnižjo plastjo (slika 5.2).

5.2.3 Primer uporabe

Uporaba knjižnice jClouds za izpis seznama računskih enot na platformi OpenStack z uporabo abstraktnih metod, ki jih knjižnica ponuja, je prikazana na primeru 5.2.

```
1  public class JCloudsNova implements Closeable {
2     private ComputeService compute;
3     private RestContext<NovaApi, NovaAsyncApi> nova;
4     private Set<String> zones;
5
6     public static void main(String[] args) {
7        JCloudsNova jCloudsNova = new JCloudsNova();
8
9        try {
10          jCloudsNova.init();
11          jCloudsNova.listServers();
12          jCloudsNova.close();
13       }
14       catch (Exception e) {
15          e.printStackTrace();
16       }
17       finally {
18          jCloudsNova.close();
19       }
20    }
21
22    private void init() {
23       Iterable<Module> modules = ImmutableSet.<Module> of(new
             SLF4JLoggingModule());
24
25       String provider = "openstack-nova";
26       String identity = "demo:demo"; // tenantName:userName
27       String password = "devstack"; // demo account uses
             ADMIN_PASSWORD too
28
29       ComputeServiceContext context = ContextBuilder.newBuilder(
             provider)
```

```
30              .endpoint("http://172.16.0.1:5000/v2.0/")
31              .credentials(identity, password)
32              .modules(modules)
33              .buildView(ComputeServiceContext.class);
34      compute = context.getComputeService();
35      nova = context.unwrap();
36      zones = nova.getApi().getConfiguredZones();
37    }
38
39    private void listServers() {
40      for (String zone: zones) {
41        ServerApi serverApi = nova.getApi().getServerApiForZone
              (zone);
42
43        System.out.println("Servers in " + zone);
44
45        for (Server server: serverApi.listInDetail().concat())
              {
46          System.out.println("  " + server);
47        }
48      }
49    }
50
51    public void close() {
52      closeQuietly(compute.getContext());
53    }
54  }
```

Listing 5.2: Uporaba knjižnice jClouds [12]

Ponovno je potrebno poudariti, da tudi tukaj lahko povsem isto programsko kodo, z minimalnimi spremembami uporabimo tudi za izpis pri katerem drugem ponudniku.

Poglavje 6

XCloud

Do sedaj smo si pogledali, kaj je računalništvo v oblaku, v kakšnih oblikah se trenutno na tržišču ponujajo storitve v oblaku, podrobneje pa sta bili opisani tudi platformi OpenStack in VmWare vCloud. Spoznali smo tudi, da se platformi med seboj precej razlikujeta in si ogledali dve obstoječi knjižnici, ki poenostavita programski dostop do obeh.

Pri obeh knjižnicah LibCloud in jClouds smo opazili, da imata kljub tem, da sta zgrajeni na povsem različnih programskih jezikih, relativno podobno arhitekturo, vendar pa si podrobne zgradbe knjižnic nismo ogledali. V tem poglavju in njegovih podpoglavjih si bomo pogledali, kako so knjižnice namenjene odpravljanju razlik med programskimi API-ji različnih platform zgrajene, kako delujejo in kako se jih uporablja. Uporaba knjižnic bo praktično prikazana.

Čeprav sta razvijalcem na voljo precej kvalitetni knjižnici za enostavno in transparentno delo z oblakom, pa smo začutili potrebo po nečem enostavnejšim in v nekaterih pogledih malenkost zmogljivejšim. Obe knjižnici predvidevata, da razvijalec ve in pozna, kakšno konfiguracijo potrebuje in pri katerem ponudniku želi postaviti računsko enoto. Neredki pa so primeri, ko razvijalec nima določenih preferenc glede ponudnika gostovanja in želi v tistem trenutku najbolj ugodno ali primerno konfiguracijo računske enote. Za takšne potrebe se zgornje knjižnice ne izkažejo najbolje, saj ne omogočajo

enostavnega poizvedovanja po ponudbah.

Ker želimo poenostaviti izbor kot tudi samo izbiro oblaka in konfiguracije bomo razvili povsem novo knjižnico, ki bo nam in ostalim razvijalcem skrila razlike med platformami in prinesla dodaten nabor funkcionalnosti. Knjižnico smo poimenovali XCloud, omogočala pa bo enostavno upravljanje z računskimi enotami platform OpenStack ter VmWare vCloud. Dodatno bomo razvili še modul, namenjen poizvedbam SLA - *Service Level Agreement*, ki bo razvijalcem olajšal izbiro ponudnika in konfiguracije sistema. Odločitev za razvoj povsem nove knjižnice se morda na prvi pogled sliši pretirana, vendar pa bomo poleg same razširitve z dodatnim modulom SLA, knjižnico tudi precej poenostavili za uporabo.

Service Level Agreement ali na kratko SLA je dogovor med ponudnikom in najemnikom storitev v oblaku o ceni, dostopnosti in konfiguraciji računske enote. Je torej nekakšna pogodoba, ki zagotavlja kvaliteto izbrane storitve. Knjižnica XCloud bo poskrbela, da bo razvijalec glede na svoje zahteve dobil seznam SLA-jev različnih ponudnikov, ki bodo ustrezali zahtevanim pogojem. Med vsemi ponudbami ima razvijalec možnost programsko ali ročno izbrati takšno, ki zagotavlja največji izkoristek za zahtevano ceno, ali pa takšno, ki zadostuje minimalnim pogojem.

Knjižnico XCloud bomo razvili podobno, kot so to storili razvijalci knjižnice jClouds v programskem okolju Java, vendar bomo poskušali ohraniti preprostost programske kode, kot je to uspelo razvijalcem knjižnice LibCloud. Za konec bomo poleg programske knjižnice storitve v oblaku predstavili tudi spletni vmesnik, preko katerega lahko tako razvijalci kot vodstveno osebje enostavno ustvarijo novo računsko enoto na različnih platformah, ali pa le hitro preverijo stanje obstoječih računskih enot.

6.1 Struktura projekta

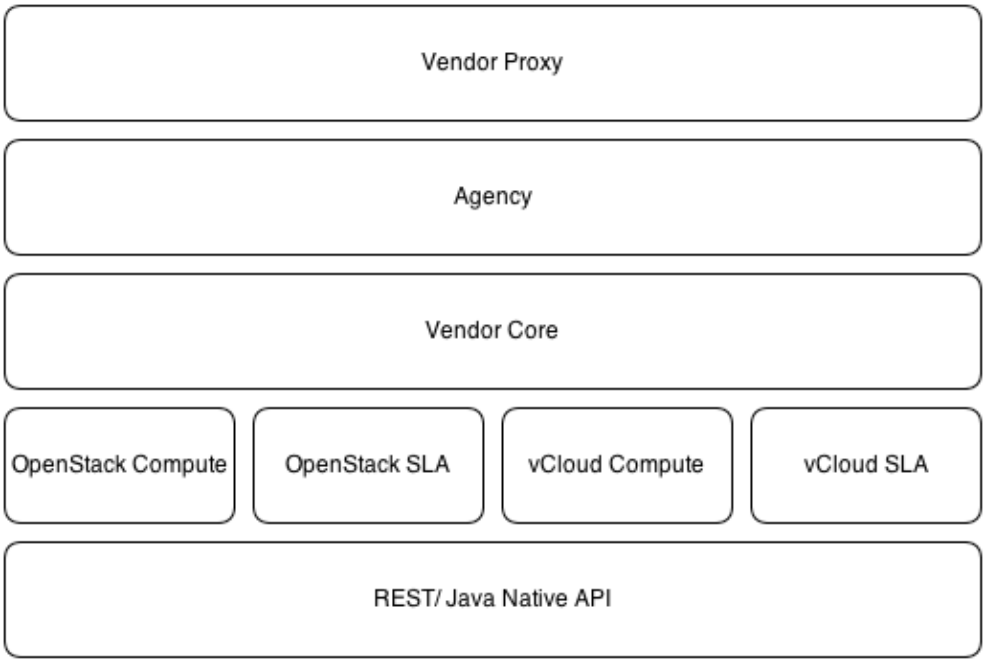
Knjižnico XCloud bomo razdelili v več medsebojno odvisnih enot zaradi boljšega razločevanja delov programske kode in lažjega razvoja posameznih komponent v prihodnost. Začrtana razdelitev enot je sledeča:

1. **XCloud Core** – matična enota, ki združuje vse enote v celoto
2. **XCloud Vendor Core** – enota, v kateri so realizirani osnovni gradniki, skupni vsem ponudnikom in definicije abstraktnih razredov
3. **XCloud OpenStack Compute Module** – implementacija abstraktnih metod podprtih funkcionalnosti za platformo OpenStack
4. **XCloud OpenStack SLA Compute Module** - implementacija SLA-jev za platformo OpenStack
5. **XCloud VmWare Compute Module** – implementacija abstraktnih metod in podprtih funcionalnosti za platformo VmWare vCloud
6. **XCloud VmWare SLA Compute Module** – implementacija SLA-jev za platformo VmWare vCloud

Struktura knjižnice je razdeljena tako, da vsaka enota sestavlja neko zaokroženo celoto. Takšna razdelitev omogoča sočasno delo več razvijalcev na ločenih komponentah.

Arhitektura knjižnice je prikazana na sliki 6.1. Razvidno je, da je tudi naša arhitektura zelo podobna preostalim knjižnicam.

V naslednjih podpoglavjih bomo z namenom boljšega razumevanja knjižnice XCloud in ostalih podobnih knjižnic podrobneje razdelali večino od šestih naštetih enot ter samo arhitekturo kot tudi implementacijo knjižnice.



Slika 6.1: Arhitektura knjižnice XCloud.

6.2 XCloud Core

Enota XCloud Core vsebuje osnovne značilnosti in osnovno strukturo projekta. Preko razredov, ki se nahajajo v tej enoti, razvijalec komunicira z različnimi tipi oblakov. Celoten projekt vsebuje le dva osnovna razreda, ki združujeta vso programsko kodo, razvito v ostalih komponentah knjižnice.

6.2.1 Agency

XCloud Agency je prvi od dveh razredov, ki prebiva v core projektu. Agency skrbi za registracijo in povezavo do vseh podprtih API dostopnih točk različnih platform. To pomeni, da je za dodajanje podpore novemu ponudniku storitev v oblaku v tem razredu potrebno registrirati ustrezni gonilnik. V primeru 6.1 je prikazana osnovna struktura razreda.

```
1
2  public class Agency {
3
4    public static final String VMWARE_ARCTUR = "VMWARE,ARCTUR";
5    public static final String OPENSTACK_GENERIC = "OPENSTACK,
         GENERIC";
6    private ArrayList<BaseComputeModule> computeModules;
7    private ArrayList<BaseSLAComputeModule> slaComputeModules;
8
9    private void initVendors() {
10     ...
11   }
12
13   public BaseComputeModule getComputeModule(String vendorName)
         throws ComputeOperationException {
14     ...
15   }
16
17   public BaseSLAComputeModule getSLAComputeModule(String
         vendorName) throws SLAException {
18     ...
19   }
20
21   public boolean setCredentials(HashMap<String, String>
         credentials)
22       throws VendorConnectionException {
23     ...
24   }
25
26   public List<BaseSLAComputeModule> getAllComputeSLAVendors() {
27     ...
28   }
29 }
```

Listing 6.1: Zgradba razreda Agency

Razred vsebuje dva seznama – enega za seznam podprtih ponudnikov računskih enot, drugega za seznam podprtih modulov SLA računskih enot.

Metoda initVendors() poskrbi za inicializacijo in registracijo podprtih ponudnikov. Novega ponudnika registriramo tako, da dodamo novo vrednost v pripadajoči seznam, kot lahko vidimo na primeru primer 6.2.

```
1 computeModules.add(new VmWareComputeModule(VMWARE_ARCTUR));
```

Listing 6.2: Registracija gonilnika ponudnika

Naslednji dve metodi *getComputeModule* in *getSLAComputeModule* poskrbita za vračanje ustreznega modula – klice nad obema metodama izvaja *Vendor Proxy*. Tudi knjižnica XCloud omogoča uporabniku, da uporablja specifične klice posamezne platforme poleg ponujenih abstraktnih programskih vmesnikov, tako kot jClouds.

Zadnji dve metodi, ki smo jih določili v razredu Agency, poskrbita za vračanje seznama vseh modulov SLA in prijavo v sistem izbranega ponudnika.

6.2.2 Vendor Proxy

Razred Vendor Proxy je vstopna točka za razvijalca, ki uporablja knjižnico XCloud. Preko njega se vrši vsa komunikacija med razvijalcem in izbranimi platformami.

Razred implementira metode vmesnikov (ang. Interface) *IComputeModule* in *IComputeSlaModule*, ki si jih bomo podrobneje ogledali v poglavju 6.3.
VendorProxy in posledično celotna knjižnica XCloud omogočata sledeče funkcionalnosti:

- ustvarjanje nove instance računske enote,
- pridobitev trenutnega stanja računske enote (zagnana, ustavljena, itd.),
- upravljanje z računskimi enotami (zaustavitev, vnovični zagon, odstranitev, zagon),
- pridobitev seznama vseh računskih enot,

- pridobitev osnovnega gonilnika za izbrano platformo (dostop do naprednih funkcionalnosti),
- generiranje zasebnega SSH ključa,
- pridobitev seznama vseh SLA-jev,
- pridobitev seznama vseh SLA-jev, ki zadostujejo nekim minimalnim pogojem - SLA vpogled.

6.3 XCloud Vendor Core

Srce celotne knjižnice se skriva v enoti Vendor Core, v kateri smo realizirali vse objekte, potrebne za manipulacijo z različnimi platformami in različne vmesnike, ki abstrahirajo razlike platform. Programska koda je organizirana v projektu Vendor Core, in sicer v paketih (ang. packages) javanskega razvojnega okolja. Oglejmo si začrtano organizacijo kode:

- *commons* – v tem paketu se nahaja skupna programska koda,
- *compute.commons* – objekti, potrebni za delovanje in upravljanje knjižnice,
- *compute* – programski vmesniki in osnovna implementacija abstrahiranega uniformnega API-ja,
- *compute.sla.commons* – objekti, potrebni za delovanje in upravljanje modula SLA,
- *compute.sla* – programski vmesniki in osnovna implementacija abstrahiranega modula SLA,
- *exceptions* – objekti, ki predstavljajo možne izjeme (ang. exceptions) med izvajanjem klicev knjižnice.

6.3.1 Modul SLA

Modul SLA je posebnost knjižnice XCloud, saj ga knjižnici, kot sta LibCloud in jClouds, ne implementirata. Modul SLA je uporabnen takrat, ko želi razvijalec čim hitreje in s čim manj napora ustvariti računsko enoto s primerno konfiguracijo. Za podporo delovanju modula SLA je potrebno definirati dva osnovna razreda:

- **ComputeCFP** – (CFP – Call For Proposal)
- **ComputeSLA** - (SLA – Service Level Agreement)

Objekta sta si med seboj zelo podobna, v ComputeCFP razvijalec specificira minimalno in maksimalno dovoljene lastnosti sistema, ki ga želi – npr. minimalno potreben RAM ter maksimalno potreben RAM itd., v objektu ComputeSLA pa razvijalec prejme dejanske ponudbe različnih ponudnikov – vsako v ločenem ComputeSLA objektu. V ComputeSLA objektu razvijalec izve točno določene vrednosti sistema, ki mu jih ponudnik storitve ponuja in seveda tudi zahtevano ceno za najem.

Logiko za delovanje modula SLA smo zastavili in realizirali v razredu *BaseSlaComputeModule*, ki implementira vmesnik, določen v *IComputeSlaModule*. Zgradbo vmesnika prikazuje primer 6.3.

```
1  public interface IComputeSlaModule {
2
3    /**
4     * Returns all available offers
5     *
6     * @return
7     */
8    public List<ComputeSLA> getAllAvailableComputeOffers()
9      throws SLAException;
10
11   /**
12    * Returns only offers that match the cfp
13    *
14    * @param cfp
```

```
15      * @return
16      */
17     public List<ComputeSLA> lookupComputeOffers(ComputeCFP cfp)
18       throws SLAException;
19
20     /**
21      * Sets credentials and establishes connection to vendor
22      *
23      * @param credentials
24      * @throws VendorConnectionException
25      */
26     public boolean setCredentials(HashMap<String, String>
           credentials)
27       throws VendorConnectionException;
28
29     /**
30      * Returns true if requires credentials to
31      * provide SLA's.
32      *
33      * @return
34      */
35     public boolean requiresCredentials();
36
37 }
```

Listing 6.3: Vmesnik ComputeSLA modula

Poudariti je potrebno, da večino trenutnih vrednosti objektov SLA knjižnica pridobi dinamično s pomočjo programskega vmesnika posameznega oblaka. V nekaterih primerih pa smo zaradi specifike posameznih ponudnikov določene vrednosti kot so cena najema določili ročno v konfiguracijskih datotekah v formatu JSON. Te datoteke in posledično polja določenih vrednosti je možno poljubno spreminjati brez posegov v samo programsko kodo knjižnice.

6.3.2 Modul Compute

Modul Compute vsebuje osnovne gradnike, ki so potrebni za delovanje celotne knjižnice, saj smo v tem trenutku v XCloud podprli le funkcionalnosti za upravljanje z računskimi enotami.

Osnovni objekt, s katerim upravljamo v okviru računskih enot, je strežnik (ang. Server), ki smo ga tudi abstrahirali in predstavili z objektom *Server*. Objekt hrani osnovne lastnosti, kot so ime, lokacija, IP naslovi itd. Za abstrakcijo API-ja za upravljanje z enotami različnih ponudnikov storitev v oblaku smo izbrali naslednjo podmnožico funkcionalnosti:

- ustvari enoto,
- zaženi enoto,
- ustavi enoto,
- vnovični zagon enote,
- odstrani enoto,
- generiraj privatni ključ za SSH prijavo,
- pridobi stanje strežnika,
- pridobi seznam vseh strežnikov.

Te funkcionalnosti smo začrtali v vmesniku *IComputeModule*, od ponudnika neodvisne metode pa realizirali v razredu *BaseComputeModule*, ki implementira vmesnik iz primera 6.4.

```
1 public interface IComputeModule {
2
3   public boolean setCredentials(HashMap<String, String>
        credentials)
4       throws VendorConnectionException;
5
6   public Server createVM(HashMap<String, String> parameters)
7       throws CreateResourceException;
```

```
8
9   public Server createVM(ComputeSLA sla) throws
        CreateResourceException;
10
11  public boolean startVM(Server server) throws
        ComputeOperationException;
12
13  public boolean stopVM(Server server) throws
        ComputeOperationException;
14
15  public boolean restartVM(Server server) throws
        ComputeOperationException;
16
17  public boolean deleteVM(Server server) throws
        ComputeOperationException;
18
19  public String generatePrivateKey(HashMap<String,String> params
        ) throws ComputeOperationException;
20
21  public ServerState getServerState(Server server) throws
        ComputeOperationException;
22
23  public HashMap<String, String> getCredentials(String vendor)
        throws ComputeOperationException;
24
25  public List<Server> getAllServers(HashMap<String, String>
        params) throws ComputeOperationException;
26
27 }
```

Listing 6.4: Vmesnik za upravljanje z računskimi enotami

Kot je razvidno iz podpisa metod vmesnika *IComputeModule*, lahko računsko enoto kreiramo na dva različna načina – preko izbrane ponudbe SLA ali z lastno določenimi parametri. Oba primera si bomo pogledali v nadaljevanju, kjer bomo prikazali delovanje knjižnice skozi nekaj praktičnih primerov.

6.4 Realizacija podpore izbrani platformi

V prejšnjih poglavjih smo si pogledali, kako je potrebno razdeliti enote, strukture in vmesnike za delovanje knjižnice. Sedaj, ko razumemo, kako so knjižnice za uniformni dostop do platform različnih ponudnikov zgrajene, si na kratko poglejmo še, kako dejansko dodati podporo nekemu ponudniku. V okviru knjižnice smo že realizirali podporo za platformi OpenStack ter VmWare, njune podrobne implementacije pa ne bomo obravnavali, saj za razumevanje delovanja nista ključni.

Če želimo dodati podporo novemu ponudniku, moramo po zaslugi preproste in učinkovite arhitekture knjižnice XCloud le razširiti (ang. Extend) razred *BaseComputeModule* in implementirati ustrezne metode za kreiranje, brisanje in upravljanje z računskimi enotami, kot prikazuje primer 6.5.

```
1 public class OpenstackComputeModule extends BaseComputeModule {}
```

Listing 6.5: Dodajanje podpore novemu ponudniku

Podobno moramo za dodajanje podpore modulu SLA izbranega ponudnika razširiti le razred *BaseSLAComputeModule*, kot je prikazano na primeru 6.6.

```
1 public class OpenStackSLAComputeModule extends
    BaseSLAComputeModule {}
```

Listing 6.6: Dodajanje SLA podpore novemu ponudniku

Od tod dalje mora razvijalec znotraj metod, ki so določene v vmesnikih *IComputeModule* in *IComputeSlaModule*, uporabiti ustrezne klice, specifične za izbranega ponudnika. Knjižnica XCloud ponuja podporo platformi OpenStack s pomočjo javanske knjižnice *OpensStack Java SDK*, platformi vCloud pa preko uradno podprtega SDK-ja. Prav tako lahko znotraj razreda, ki razširja funkcionalnosti *BaseComputeModul*-a, razvijalec doda tudi za ponudnika značilne metode, za katerega razvija podporo in tako zagotovi podporo naprednim funkcionalnostim poleg tistih, ki jih določa že abstrahiran nabor.

6.5 Uporaba knjižnice

Uporaba knjižnice XCloud je preprosta in ne zahteva veliko dodatnega učenja s strani razvijalca. Del preprostosti gre seveda pripisati tudi omejenemu naboru funkcionalnosti, ki jih knjižnica trenutno podpira, vendar je nabor funkcionalnosti približno enak tistim, ki nam jih ponuja LibCloud – torej dovolj velik, da je že praktično uporaben.

6.5.1 Inicializacija in povezava z ponudnikom

```
1 VendorProxy proxy = new VendorProxy();
2 HashMap<String, String> cred = new HashMap<String, String>();
3 cred.put(XParams.VENDOR, Agency.VMWARE_ARCTUR);
4 cred.put(XParams.VmWare.USERNAME_FIELD, "USERNAME");
5 cred.put(XParams.VmWare.PASSWORD_FIELD", "PASSWORD");
6 cred.put(XParams.VmWare.ADMIN_URL_FIELD, "https://vcloud.arctur.
      si");
7
8 proxy.setCredentials(cred)
```

Listing 6.7: Povezava z izbranim ponudnikom

6.5.2 Kreiranje instance z ročno določenimi parametri

```
1 HashMap<String, String> prop = new HashMap<String, String>();
2 prop.put(XParams.VENDOR, Agency.VMWARE_ARCTUR);
3 prop.put(XParams.VmWare.ORG_PARAM, "organizacija");
4 prop.put(XParams.VmWare.VDC_PARAM, "datacenter");
5 prop.put(XParams.VmWare.CATALOG_PARAM, "CentOS");
6 prop.put(XParams.VmWare.TEMPLATE_PARAM, "CentOS-6-x64");
7 prop.put(XParams.VmWare.NETWORK_PARAM, "network");
8 prop.put(XParams.VmWare.PROJECT_PARAM, "imeprojekta");
9 Server s = proxy.createVM(prop);
```

Listing 6.8: Kreiranje računske enote

6.5.3 Kreiranje instnace z SLA vpogledom

```
1  ComputeCFP cfp = new ComputeCFP();
2  cfp.setMinCpuCores(2);
3  cfp.setMaxCpuCores(4);
4  cfp.setMinRamMB(512);
5  cfp.setMaxRamMB(2048);
6  cfp.setMaxPricePerHourUSD(0.42)
7
8  List<ComputeSLA> slas = proxy.lookupComputeOffers(cfp);
9
10 Server s = proxy.createVM(slas.get(0));
```

Listing 6.9: Kreiranje računske enote z izborom SLA

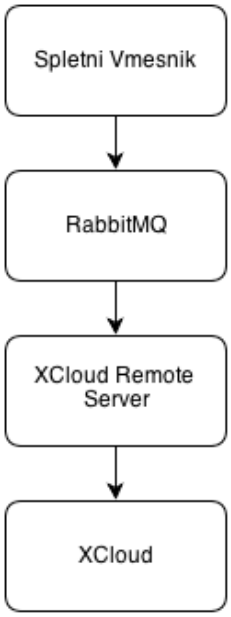
Iz primerov je razvidno da je uporaba knjižnice v praksi preprosta in večino dela ima razvijalec z izbiro in nastavitvijo parametrov za kreiranje novih računskih enot, od tod dalje pa so klici preprosti in večinoma enovrstični.

6.6 Spletni vmesnik

V okviru knjižnice XCloud smo razvili tudi spletni vmesnik, ki omogoča enoten uporabniški vmesnik za dostop do različnih ponudnikov tudi manj veščim uporabnikom, ki se ne želijo ubadati s programskimi vmesniki.

Za izgradnjo spletnega vmesnika smo izbrali ogrodje Django – programsko okolje v jeziku Python, namenjeno razvoju spletnih strani v tehnologiji MVC. Ker je knjižnica XCloud razvita v Javi, je potrebno poskrbeti tudi za prenos klicev med jezikoma – za komunikacijo med spletnim vmesnikom in knjižnico smo izbrali tehnologijo RabbitMQ – odprtokodno AMQP orodje s knjižnicami v različnih jezikih.

Spletni vmesnik podpira večino funkcionalnosti, ki jih nudi programska knjižnica XCloud.



Slika 6.2: Arhitektura spletnega vmesnika.

6.6.1 Arhitektura

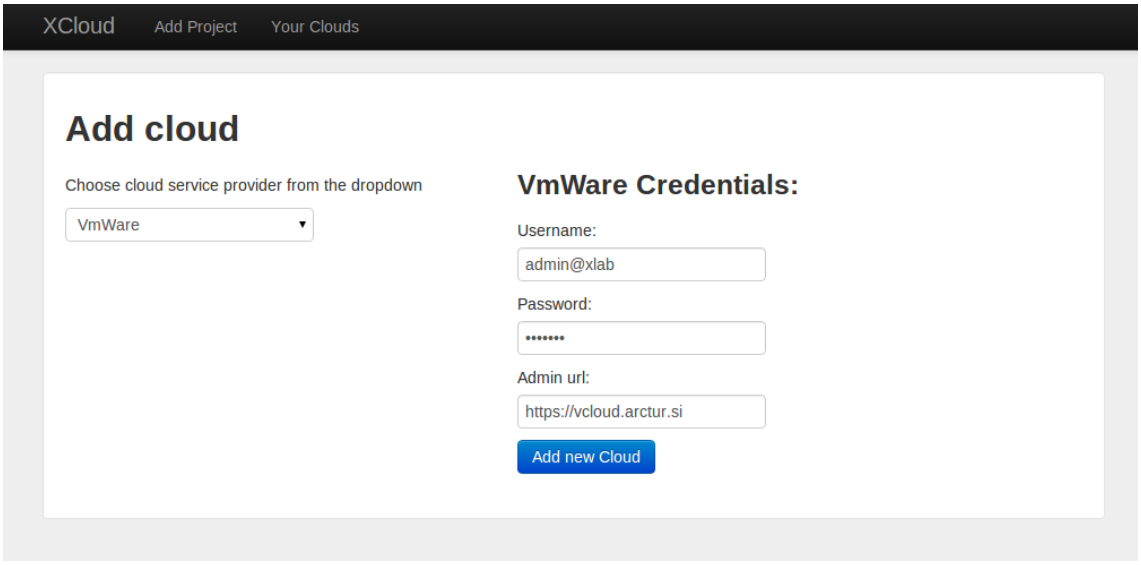
Arhitekturo zastavljenega spletnega vmesnika prikazuje slika 6.2.

Uporabniške akcije, ki se izvšijo na spletnem vmesniku, se pošljejo preko protokola AMQP - RabbitMQ do javanske komponente *XCloud Remote Server*, ki posluša za novimi sporočili in glede na posamično sporočilo proži klice v knjižnici XCloud. Odgovori in podatki o računskih enotah nato potujejo po isti poti nazaj do spletnega vmesnika.

6.6.2 Uporaba spletnega vmesnika

Za uporabo spletnega vmesnika mora uporabnik najprej vnesti svoje prijavne podatke za izbrani oblak. Spletni vmesnik prav tako podpira obe platformi - OpenStack in vCloud. Postopek dodajanja novega oblaka je prikazan na sliki 6.3.

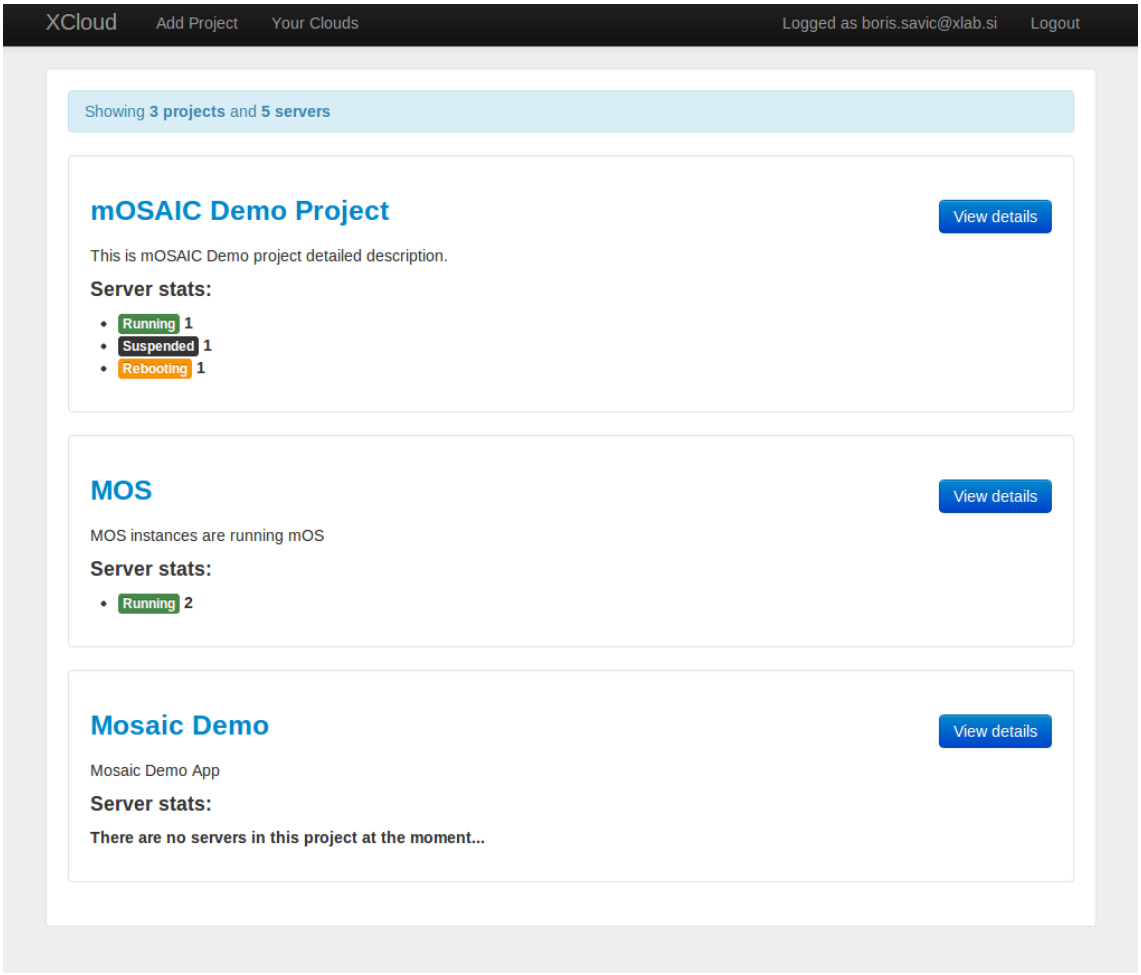
Znotraj vmesnika si razvijalec lahko ustvari projekte in s tem smiselno grupira računske enote. Slika 6.4 prikazuje domačo stran vmesnika, kjer



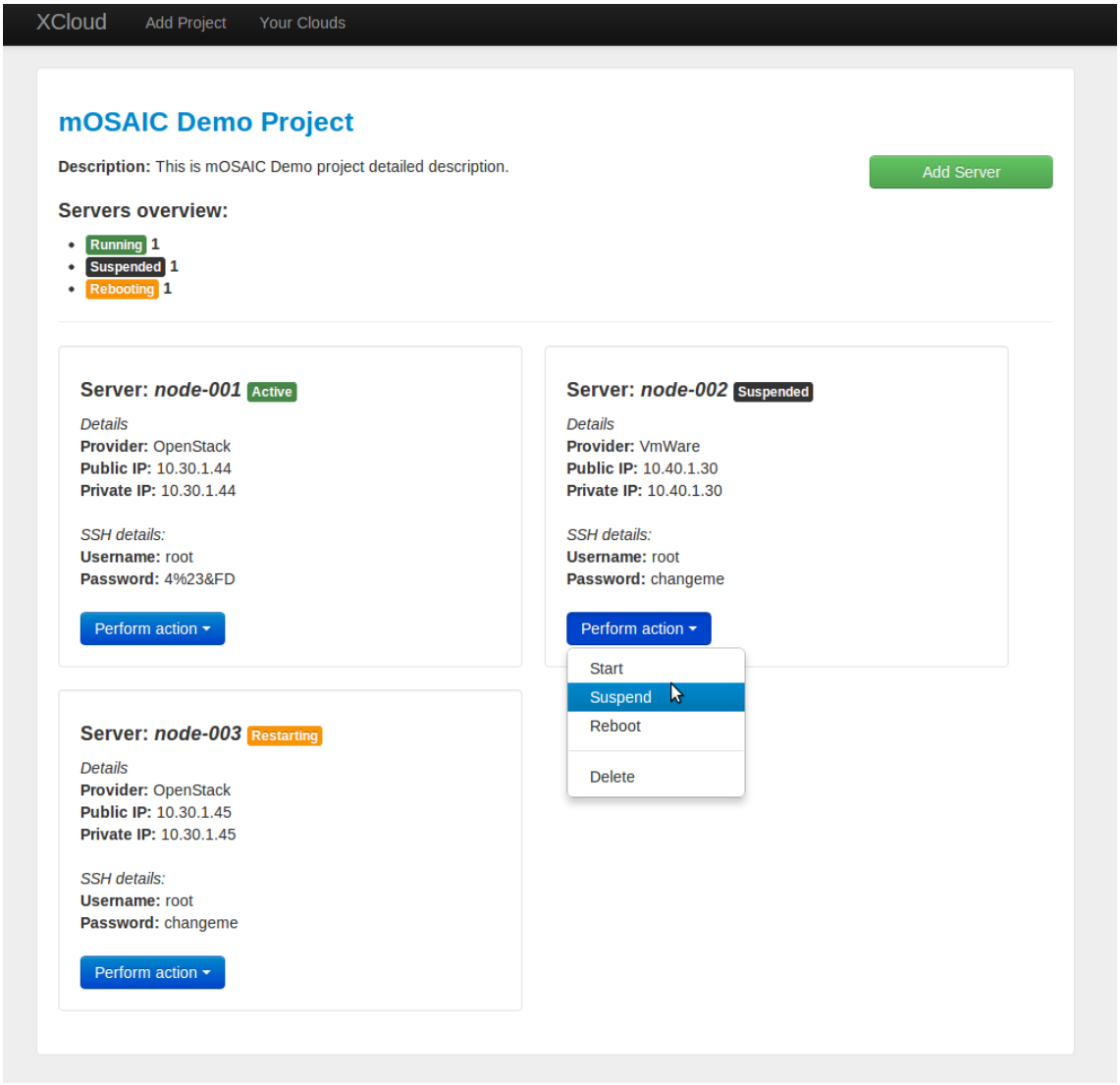
Slika 6.3: Dodajanje prijavnih podatkov za izbrano platformo

uporabnik lahko hitro pregleda stanje strežnikov posameznega projekta.

Podrobnejše stanje posameznih računskih enot lahko uporabnik pregleda na strani s podrobnostmi projekta, kakor prikazuje slika 6.5. Uporabnik ima možnost upravljanja s posameznimi računskimi enotami in enostaven ter hiter pregled nad osnovnimi podatki posameznega strežnika.



Slika 6.4: Seznam projektov/skupin računskih enot



Slika 6.5: Seznam računskih enot znotraj izbranega projekta

Poglavje 7

Zaključek

V diplomskem delu smo obravnavali računalništvo v oblaku, in sicer smo natančneje primerjali platformi OpenStack in VmWare vCloud. Spoznali smo nekaj osnovnih razlik med platformami in si pogledali, kako lahko omenjene platforme uporabljamo programsko preko API klicev. Ker se programski vmesniki različnih platform precej razlikujejo, smo spoznali dve obstoječi knjižnici za enostavnejšo uporabo različnih platform.

Dotaknili smo se tudi osnovne arhitekture tovrstnih knjižnic in ugotovili, da so si kljub različnim programskim jezikom zelo podobne in implementirajo podobne koncepte ter strukture. Za boljše razumevanje arhitekture in zaradi potrebe po dodatnih funkcionalnostih, kot so vpogled v ponudbe SLA, smo razvili tudi svojo knjižnico. Poimenovali smo jo XCloud.

Osnovna arhitektura knjižnice XCloud je prav tako zelo podobna arhikturam ostalih odprtokodnih rešitev na tržišču, vendar pa vsebuje nekaj dodatnih funkcionalnosti. Ena takšnih funkcionalnosti je SLA vpogled, ki omogoča razvijalcu preprost vpogled v ponudbe, ki jih različni ponudniki storitev v oblaku nudijo. Tako razvijalcu ni potrebno poznati specifičnih lastnosti programskih vmesnikov posamezne platforme in različnih konfiguracij sistemov, ki se razlikujejo pri različnih ponudnikih. Omogočeno pa mu je tudi, da dinamično kreira računske enote pri različnih ponudnikih, kar pomeni, da se lahko glede na pogoje, ki jih je zastavil, računska enota kreira

pri katerem koli ponudniku, ki zadosti minimalnim pogojem. Torej smo s knjižnico XCloud abstrakcijo različnih oblakov popeljali še en nivo višje, saj razvijalcu zdaj ni potrebno več skrbeti za izbiro pravega ponudnika storitev v oblaku.

Druga dodatna specifika knjižnice XCloud je njen spletni vmesnik. Spletni vmesnik smo razvili z namenom poenostaviti uporabo več oblakov tudi vodstvenemu osebju, ki morda želi le hitro in enostavno vizualno pregledati stanje računskih enot. Spletni in programski vmesnik abstrahirata razlike med oblaki in omogočata preprosto upravljanje različnih oblakov.

Če kritično pogledamo nazaj, bi morali v knjižnico vgraditi tudi podporo za vzpostavljanje SSH povezave in nameščanje programskih paketov na kreirani računski enoti, tako kot nam to omogočajo ostale knjižnice. Trenutna verzija knjižnice XCloud (še) ne podpira programskega vmesnika za upravljanje in manipuliranje z operacijskim sistemom ustvarjene računske enote, temveč omogoča le neposredno manipulacijo s samimi računskimi enotami. Knjižnici in uporabniškemu vmesniku XCloud bi lahko v prihodnosti dodali tudi podporo za upravljanje s shrambo podatkov v oblaku in podporo več različnim ponudnikom. Tovrstna nadgradnja bi bila relativno preprosta, saj je arhitektura knjižnice dovolj preprosta in modularno zastavljena, da omogoča enostavno dodajanje novih storitev in ponudnikov.

Za morebiten nadaljnji razvoj knjižnice XCloud bi bilo treba ustvariti še podporo SSH povezavi, saj bi k sami uporabnosti knjižnice veliko pripomogla. Poleg tega, bi bilo na področju modula SLA, v prihodnosti, v knjižnico smiselno vgraditi tudi mehanizme, ki bi poskrbeli za samodejno posodabljanje določenih vrednosti ponudb, saj so v trenutni implementaciji, kot smo videli, določene vrednosti vnaprej določene v konfiguracijskih datotekah.

Literatura

[1] L. Wang, G. Laszewski, M. Kunze and J. Tao, "Cloud computing: a perspective study", *J New Generation Computing*, 2010, pp 1-11.

[2] Sushil Bhardwaj, L. Jain, S. Jain, "Cloud computing: A study of infrastructure as a service(IaaS)", *International Journal of Engineering and Information Technology*, 2010

[3] Matthias Kaiserswerth and others (2012). *White Paper Cloud Computing*. Swiss Academy of Engineering Sciences

[4] (2013) OpenStack. Dostopno na: http://www.openstack.org

[5] (2013) OpenStack Foundation. Dostopno na http://www.openstack.org/foundation/companies/

[6] (2013) Wikipedia - VmWare. Dostopno na http://en.wikipedia.org/wiki/VMware

[7] (2013) Arctur. Dostopno na http://arctur.si/resitve_in_storitve/cloud_computing/

[8] (2013) VmWare vCloud. Dostopno na http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html

[9] (2013) LibCloud - About. Dostopno na http://libcloud.apache.org/about.html

[10] (2013) LibCloud - Features. Dostopno na http://libcloud.apache.org/driver-features.html

[11] (2013) LibCloud - Get Started. Dostopno na http://libcloud.apache.org/getting-started.html

[12] (2013)jClouds. Dostopno na http://jclouds.incubator.apache.org /documentation/quickstart/openstack/

[13] (2013) XCloud. Dostopno na https://bitbucket.org/boris_savic/xcloud-core