Matematički model predikcije korištenja računalnih resursa za razvojnu i testnu okolinu

Ćosić, Terezija

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Terezija Ćosić

MATEMATIČKI MODEL PREDIKCIJE KORIŠTENJA RAČUNALNIH RESURSA ZA RAZVOJNU I TESTNU OKOLINU

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, rujan, 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Rješenja u oblaku postala su sastavni dio poslovanja gotovo svih IT kompanija, što je slučaj i kod tvrtke Ericsson Nikola Tesla koja razvija virtualne okoline za testiranje i razvoj poslužiteljskih aplikacija na platformi otvorenog koda za računanje u oblaku OpenStack. Okoline se sastoje od više virtualnih strojeva, a krajnjim korisnicima - programerima i inženjerima unutar tvrtke - se dostavljaju putem projekta TEAaaS - Test Environment and Application as a Service. TEAaaS je rješenje u oblaku zasnovano, očekivano, na OpenStacku, a korisnici mu pristupaju putem web sučelja TEAaaS Cloud Web Portal.

Cilj ovog rada je mjerenje potrošnje računalnih resursa u virtualnim okolinama i njihovo spremanje u bazu podataka kako bi drugim odjelima tvrtke pružili podatke koje oni koriste u svojim istraživanjima sa svrhom izrade korisničkih profila, kao i mjerenje zauzeća diskovnog prostora na fizičkim serverima. Zanima nas kakav rast potrošnje diskovnog prostora možemo očekivati u budućnosti. Ti podaci će poslužiti u svrhu planiranja nabavka nove opreme na odjelu na kojem se i rade mjerenja.

U prvom poglavlju objašnjavamo pojam računanja u oblaku, modele upotrebe i implementacije te razloge zbog kojih su IT poduzeća prihvatila taj koncept i sve više rade s njim.

Dalje, u drugom dan je pregled MySQL sustava za upravljanje relacijskim bazama podataka koji koristimo za spremanje mjerenih podataka.

Treće poglavlje opisuje OpenStack i njegove glavne komponente uz primjer stvaranja virtualnog stroja u OpenStack virtualnoj okolini gdje se vidi uloga svake komponente i komunikacija među njima tijekom tog procesa.

Opisi mjerenih fizičkih i virtualnih okolina su dani u četvrtom poglavlju, skupa sa strukturom baze podataka u koju se mjereni podaci spremaju i programskim kodom kojim se dobivaju mjerenja.

Poglavlje 1

Računanje u oblaku

Dok još nije ni postojalo kao pojam, računanje u oblaku je odgovaralo nekim idejama iz 70-tih godina prošlog stoljeća o računarstvu budućnosti, a koje zamišljaju računarstvo kao javno dostupnu uslugu poput telefonije i telefonskih centrala (John McCarthy, 1961.) ili predviđaju veliki razvoj uslužnih programa kao posljedicu razvoja mreža računala (Leonard Kleinrock, 1969.). U nastavku izdvajamo događaje iz povijesti mrežne i računalne industrije koji su pridonijeli stvaranju računanja u oblaku kao softverskog modela i modela računalne infrastrukture, objašnjavamo osnovne pojmove računanja u oblaku te modele usluge i implementacije.

1.1 Povijesni pregled

Pojmovi "mrežni oblak" ili "oblak" uvedeni početkom 1990-ih odnose se na prijenos podataka s jedne krajnje točke (lokalne mreže) u "oblak" (širokopojasna mreža) i zatim dalje do druge krajnje točke. Preciznije, pojam potječe iz svijeta telekomunkacija kada su telekomunikacijske tvrtke počele nuditi uslugu virtualne privatne mreže (*Virtual Private Network*) s usporedivom kvalitetom usluge, a za manju cijenu. Upotrebom usluge virtualne privatne mreže promet kroz nju se mogao preusmjeriti kako bi se uravnotežila iskorištenost cijele mreže. Računanje u oblaku sada proširuje usluge virtualne privatne mreže na pokrivanje servera i mrežne infrastrukture [4].

Računalna i mrežna industrija odigrala je ključnu ulogu u implementaciji i razvoju računanja u oblaku. Salesforce.com 1999. godine uvodi pojam softvera kao usluge, detaljnije objašnjen u odjeljku 1.3. Amazon.com 2006. godine pokreće platformu Amazon Web Services (AWS), paket usluga usmjerenih na poslovanje koji omogućuje daljinski pristup pohrani, računalnim resursima i poslovnim funkcionalnostima. Središnji dio Amazon Web Services platforme čini Elastic Compute Cloud (EC2), objavljen nekoliko mjeseci nakon same platforme. Elastic Compute Cloud omogućuje korisnicima iznajmljivanje virtualna

strojeva na kojima rade vlastite računalne aplikacije. Korisnik može stvoriti, pokrenuti i terminirati virtualne strojeve prema potrebi, plaćajući sat po aktivnoj upotrebi istih. Uskoro i IBM i Google započinju istraživačke projekte na području računanja u obliku. Google Apps su danas sastavni dio aplikacijskog paketa svakog Google korisnika. S inicijalnim izdanjem 2008. godine, Eucalyptus postaje prva platforma otvorenog koda za razvoj privatnih računalnih oblaka.

Poslovne motivacije za razvoj računalnih oblaka

U nastavku navodimo glavne motive koji su potaknuli organizacije na prihvaćanje koncepta računanja u oblaku u vlastitom poslovanju, što uključuje potražnju za uslugama računanja u oblaku s jedne strane i ponudu s druge.

Planiranje kapaciteta. Planiranje kapaciteta je proces određivanja i ispunjavanja budućih zahtjeva IT resursa (hardvera ili softvera), proizvoda i usluga organizacije. U tom kontekstu, kapacitet predstavlja najveću količinu posla koju IT resurs može isporučiti u određenom vremenskom razdoblju. Razlika između kapaciteta IT resursa i njegove potražnje može dovesti do toga da sustav postaje neučinkovit (*over-provisioning*) ili ne može ispuniti potrebe korisnika (*under-provisioning*). Planiranje kapaciteta je usmjereno na minimiziranje ove razlike kako bi se postigla predvidljiva učinkovitost i performanse, što može predstavljati izazov zbog procjene oscilacija u opterećenju resursa. Postoji stalna potreba za uravnotežavanjem maksimalnih i minimalnih zahtjeva za korištenjem bez nepotrebnih pretjeranih izdataka za infrastrukturu.

Smanjenje troškova. Izravno usklađivanje IT troškova i uspješnosti poslovanja može biti teško održavati. Rast IT okruženja često odgovara procjeni maksimalnih zahtjeva za infrastrukturom jer je potencijal iskorištavanja određenog poslovnog rješenja automatizacije uvijek ograničen procesorskom snagom svoje temeljne infrastrukture. Uz troškove stjecanja nove infrastrukture potrebno je voditi računa i o troškovima vlasništva nad istom. Potonje uključuje troškove tehničkog osoblja potrebnog za održavanje IT okruženja, administrativnog osoblja zaduženog za praćenje licenci i podrške, implementacije i testiranja novih nadogradnji i zakrpa te sigurnosnih mjera i na kraju troškove samog napajanja i hlađenja infrasutrkture. Dakle, kontinuirano vlasništvo nad unutarnjom tehnološkom infrastrukturom može obuhvatiti opterećujuće odgovornosti koje nameću složene utjecaje na korporativne proračune.

Agilnost organizacije. Agilnost organizacije je mjera odaziva organizacije na promjene prouzročene unutarnjim i vanjskim faktorima. IT poduzeće često treba odgovoriti na promjenu poslovanja skaliranjem IT resursa izvan dosega prethodno predviđenog

ili planiranog. Ulaganja i troškovi vlasništva nad infrastrukturom potrebni za omogućavanje novih ili proširenih rješenja za automatizaciju poslovanja mogu biti dovoljna prepreka za poduzeće za zadovoljavanje ne potpuno odgovarajućom IT infrastrukturu, čime se smanjuje sposobnost poduzeća da odgovori zahtjevima stvarnog svijeta, održi korak s tržišnim zahtjevima, konkurentskim pritiscima i vlastitim strateškim poslovnim ciljevima.

Iz navedenih razloga vidljivo je zašto se poduzeća odlučuju na korištenje raznih rješenja "iz oblaka", a svi su uglavnom financijske prirode. Svojevrsni najam IT infrastrukture i IT okruženja te fleksibilnost tog pristupa pružaju poduzećima veću slobodu upravljanja investicijama i troškovima.

Tehnološki preduvjeti razvoja računanja u oblaku

U nastavku navodimo tehnološke inovacije koje su poslužile kao implementacijski i idejni temelj računanju u oblaku.

Klaster. Klaster je skupina nezavisnih IT resursa koji su međusobno povezani i funkcioniraju kao jedan sustav pri čemu se stope kvara sustava smanjuju, dok se dostupnost i pouzdanost povećavaju. Opći preduvjet hardverskog klasteriranja je da njegove komponente imaju razumno identičan hardver i operacijske sustave što osigurava slične performansi kada jedna neuspjela komponenta treba zamijeniti drugo. Komponentni uređaji koji tvore skupinu održavaju se u sinkronizaciji putem za to određenih brzih komunikacijskih veza. Osnovni koncept ugrađene redundancije i preuzimanja posla u slučaju kvara temeljni su za platforme u oblaku.

Računalna rešetka (*Grid Computing*). Računalna rešetka pruža platformu u kojoj su računalni resursi organizirani u jedan ili više logičkih skupova. Ovi se skupovi kolektivno koordiniraju kako bi pružili distribuiranu mrežu visokih performansi, koja se ponekad naziva "super virtualnim računalom". Računalna rešetka se razlikuje od klasteriranja u tome da su elementi računalne rešetke distribuiriani i labavije spojeni. Kao rezultat toga, računalna rešetka može uključivati računalne resurse koji su heterogeni i zemljopisno raspršeni, što općenito nije moguće s računalnim sustavima organiziranima u klaster.

Virtualizacija. Virtualizacija predstavlja tehnološku platformu koja se koristi za stvaranje virtualnih instanci IT resursa. Slojevi softvera za virtualizaciju omogućuju fizičkim IT resursima pružanje više svojih virtualnih slika, tako da njihove temeljne mogućnosti može dijeliti više korisnika. Bez virtualizacije, softver je ograničen "na boravak" i povezivanje sa statičkim hardverskim okruženjima. Proces virtualizacije

otklanja tu softversku ovisnost o hardveru jer hardverski zahtjevi mogu biti simulirani emulacijskim softverom koji se izvodi u virtualiziranim okruženjima.

Bitno je još istaknuti i neke druge elemente koji su omogućili razvoj računanja u oblaku - širokopojasne mreže, razvoj podatkovnih centara i web tehnologije.

1.2 Osnovni pojmovi

Prema definiciji iz [2], **računanje u oblaku** je specijalizirani oblik distribuiranog računarstva koji uvodi uslužne modele za pružanje skalabilnih i mjerenih resursa na daljinu.

Navodimo neke osnovne pojmove iz područja računanja u oblaku, a koje ćemo koristiti u idućim odjeljcima.

Oblak. Oblak se odnosi na različite IT okoline dizajnirane radi pružanja skalabilnih i mjerenih IT resursa na daljinu. IT resursi unutar oblaka imaju svrhu podržati *back-end* procese i omogućiti korisnički pristup istima. U softverskom inženjerstvu *back-end* se odnosi na podatkovni pristupni sloj koji u kojem je sadržana logika softvera i brine o pohrani podataka. Oblak se ne temelji nužno na webu (internetskim protokolima i tehnologijama), iako to obično je slučaj. Protokoli se odnose na standarde i metode kojima računala međusobno komuniciraju na unaprijed definiran i strukturiran način, tako da se oblak može temeljiti na korištenju bilo kojeg protokola koji omogućuje daljinski pristup IT resursima.

Pružatelj i potrošač oblaka (*cloud provider, cloud consumer*). Stranka koja pruža IT resurse temeljene na oblaku je **pružatelj oblaka**. Stranka koja koristi IT resurse temeljene na oblaku je **potrošač oblaka**. Ovi pojmovi predstavljaju uloge koje obično preuzimaju organizacije u odnosu na oblak i odgovarajuće ugovore o pružanju usluga u oblaku.

Usluga u oblaku i potrošač usluge u oblaku. Iako je oblak okolina kojoj je omogućen pristup na daljinu, svi IT resursi unutar oblaka ne moraju moraju također biti dohvatljivi. Na primjer, baza podataka ili fizički poslužitelj koji se distribuira unutar oblaka mogu biti dostupni samo drugim IT resursima unutar istog oblaka. Kako bi se omogućio pristup udaljenim klijentima može se posebno koristiti softverski program s objavljenim aplikacijskim programskim sučeljem. **Usluga u oblaku** je tako bilo koji IT resurs kojem se pristupa na daljinu putem oblaka. **Potrošač usluge u oblaku** je privremena uloga koju preuzima softverski program kada pristupa usluzi u oblaku.

1.3 Modeli usluga računanja u oblaku

Model usluge računanja u oblaku predstavlja specifičnu, predefiniranu kombinaciju IT resursa koju nudi pružatelj oblaka. NIST (National Institute of Standards and Technology) definira tri uobičajena modela usluga računanja u oblaku koja navodimo i objašnjavamo u nastavku [6].

Infrastruktura kao usluga (*Infrastructure as a Service - IaaS*). Ovaj model predstavlja samostalnu IT okolinu sastavljeno od IT resursa kojima se može pristupiti i upravljati putem sučelja i alata temeljenih na uslužnim programima za oblak. Navedena okolina može uključivati hardver, mrežu, operativne sustave i ostale "sirove" IT resurse. IT resursi su obično virtualizirani i zapakirani u pakete koji pojednostavljuju prilagodbu infrastrukture, a primarni IT resursi unutar tipičnog IaaS okruženja su virtualni poslužitelj. Opća svrha IaaS modela je pružanje potrošačima oblaka visoku razinu kontrole i odgovornosti nad konfiguracijom i upotrebom istog.

Platforma kao usluga (*Platform as a Service - PaaS*). Potrošač ima mogućnost na infrastrukturi oblaka implementirati aplikacije stvorene pomoću programskih jezika, biblioteka, usluga i alata koje podržava pružatelj oblaka. Korisnik ne upravlja, ali ima kontrolu nad implementiranim aplikacijama i postavkama konfiguracije za okruženje u kojem radi.

Uobičajeni razlozi zbog kojih bi potrošač oblaka odabrao PaaS model su:

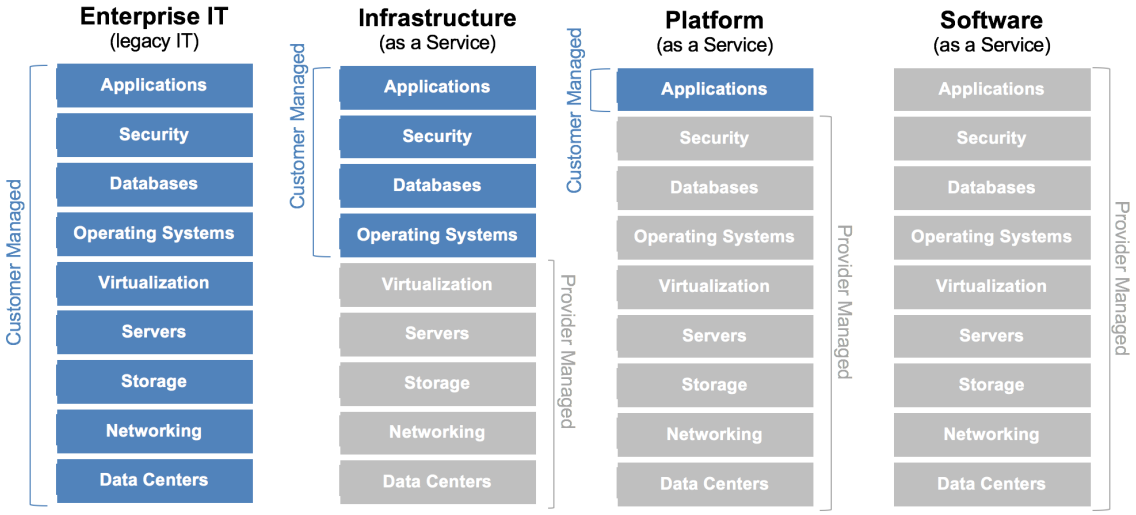
- proširenje *on-premise* okoline (okolina unutar granica poduzeća koja nije bazirana na oblaku) na oblak radi skalabilnosti i ekonomskih razloga
- potpuna zamjena *on-premise* okoline onom u oblaku
- korisnik želi postati pružatelj usluga u oblaku razvojem vlastitih usluga u danoj okolini

Radom u sklopu gotove platforme, potrošač oblaka se poštedio administrativnog opterećenja postavljanja i održavanja golemih infrastrukturnih resursa koji se pružaju putem IaaS modela. S druge strane, potrošač oblaka dobiva nižu razinu kontrole nad temeljnim IT resursima u odnosu na IaaS model.

Softver kao usluga (*Software as a Service - SaaS*). Potrošač ima mogućnost korištenja aplikacija pružatelja usluga koje se izvode na infrastrukturi oblaka, a može im se pristupiti s različitih klijentskih uređaja kao što je web preglednik ili sučelje programa. Korisnici ne upravljaju osnovnom infrastrukturom oblaka, što uključuje mrežu, poslužitelje, operacijske sustave, pohranu ili čak pojedinačne mogućnosti

aplikacije, uz moguću iznimku ograničenih korisničkih postavki konfiguracije aplikacija. Ovaj model se obično koristi za izradu obnovljive usluge oblaka (često komercijalne) dostupne širokom rasponu potrošača oblaka. Vlasnik usluge je najčešće pružatelj oblaka, no ne nužno. Na primjer, organizacija koja djeluje kao potrošač u oblaku tijekom rada u PaaS modelu može kreirati uslugu za oblak koju odluči implementirati u istom okruženju SaaS ponudu.

Razlike među modelima vidljive su i na slici 1.1.



Slika 1.1: Razlike među modelima usluga računanja u oblaku

1.4 Modeli izvedbe oblaka

Model izvedbe oblaka predstavlja poseban oblik okoline u oblaku. Neovisno o modelima pružanja usluga postoje četiri različita modela izvedbe koji se razlikuju po vlasništvu, veličini i pristupu, a navodimo ih u nastavku.

Javni oblak. Javni oblak je javno dostupna okolina u oblaku u vlasništvu pružatelja oblaka treće strane. Servisi javnog oblaka dostupni su preko javno dostupne mreže. Javni oblaci mogu biti besplatni, nuditi se za određenu cijenu ili su komercijalizirani putem drugih načina, primjerice oglasa. Pružatelj oblaka je odgovoran za stvaranje i održavanje javnog oblaka i pripadnih IT resursa.

Privatni oblak. Privatni oblak je vlasništvo jedne organizacije, a omogućuje organizaciji upotrebu tehnologije računanja u oblaku kao sredstvo za centraliziranje pristupa informatičkim resursima na različitim odjelima ili lokacijama organizacije. Organizacije koriste privatne oblake kada trebaju veći nadzor nad podacima nego što ga mogu imati korištenjem javnog oblaka. Privatnim oblakom može upravljati sama organizacija ili netko drugi. U prvom slučaju, ista organizacija je istodobno i pružatelj i davatelj oblaka. Kako bi se ove uloge razlikovale zasebni odjel preuzima zadatak pružatelja oblaka, dok svi drugi odjeli imaju ulogu potrošača oblaka.

Zajednički oblak. Zajednički oblak sličan je javnom oblaku, osim što mu je pristup ograničen na određenu skupinu (zajednicu) korisnika. Može biti u zajedničkom vlasništvu nekoliko organizacija koje dijele odgovornost održavanja ili u vlasništvu pružatelja oblaka treće strane koji pruža javni oblak s ograničenim pristupom.

Hibridni oblak. Hibridni oblak je okolina koja se sastoji od dva ili više različitih modela implementacije oblaka. Na primjer, potrošač oblaka može odlučiti implementirati usluge obrade osjetljivih podataka na privatni oblak, a manje osjetljivih u javni oblak. Mogućnost proširivanja privatnog oblaka s resursima javnog oblaka može se koristiti za održavanje razine uslužnih servisa kako bi se lakše izdržala velika opterećenja. Hibridna implementacijska arhitektura može biti složena i izazovna za stvaranje i održavanje zbog potencijalnih razlika u okolinama u oblaku i činjenice da su zadaci upravljanja oblakom obično podijeljeni između pružatelja javnog oblaka i pružatelja privatnog oblaka.

Poglavlje 2

MySQL

2.1 Povijesni pregled

Sustav za upravljanje bazom podataka (*Data Base Management System - DBMS*) je poslužitelj baze podataka. On oblikuje fizički prikaz baze u skladu s traženom logičkom strukturom, u ime klijenata obavlja sve operacije s podacima, brine o sigurnosti podataka te automatizira administrativne poslove s bazom [5].

Sustav za upravljanje relacijskom bazom podataka (*Relational database management system - RDBMS*) je bitan alat u mnogim područjima - poslovanju, istraživanjima, obrazovanju i kao podrška aplikacijama od kojih ćemo izdvojiti pretraživačke mehanizme Internet tražilica. No, unatoč važnosti dobrog sustava za upravljanje informacijskim resursima mnoge organizacije si ga jednostavno nisu mogle priuštiti. Povijesno gledano, sustavi baza podataka su bili skupa stavka u poslovanju jer su dobavljači naplaćivali prilične naknade i za sam softver i za podršku. Isto tako, budući da su mehanizmi baza podataka često zahtijevali znatne hardverske resurse za rad s razumnim performansama, cijena je bila još veća. Danas je situacija drugačija i na softverskoj i na hardverskoj strani pošto gotovo svatko od nas posjeduje osobno računalo na kojem je moguće pokrenuti besplatni operacijski sustav kao što je, na primjer, bilo koja Linux distribucija.

Upravo su projekti slobodnog koda, gdje su neki od rezultata i spomenute Linux distribucije, omogućili i široku dostupnost softverskih sustava za baze podataka. Jedan od takvih sustava je i MySQL, SQL klijent/poslužitelj za upravljanje relacijskom bazom podataka. Potječe iz Skandinavije, a uključuje SQL poslužitelj, klijentske programe za pristup poslužitelju, administrativne alate i programsko sučelje za pisanje vlastitih programa.

Korijeni MySQL-a započinju 1979. godine s UNIREG alatom za baze podataka kojeg je stvorio Michael "Monty" Widenius za švedsku tvrtku TcX. 1994. godine TcX je započeo potragu za sustavom za upravljanje relacijskom bazom podataka koji bi imao SQL sučelje za korištenje u razvoju web aplikacija. Testirani komercijalni poslužitelji su bili prespori

za TcX-ove velike tablice. U obzir je uzet i mSQL, ali nije imao određene značajke koje je zahtijevao TcX, slijedom čega je Monty počeo razvijati novi poslužitelj. To je rezultiralo novim SQL sučeljem, ali gotovo istim programskim sučeljem kao kod mSQL-a jer su za mSQL dostupni neki besplatni alati koji se uz manje izmjene mogu koristiti i za MySQL. Tako je stvoren MySQL - besplatni sustav za upravljanje bazom podataka.

Iduće godine (1995.), David Axmark započinje proces objavljivanja MySQL-a na internetu. Tako je 1996. godine MySQL 3.11.1 postao dostupan u obliku binarnih distribucija za Linux i Solaris. Danas MySQL radi na mnogim drugim platformama, od komercijalnih operacijskih sustava do poslovnih poslužitelja i dostupan je u binarnom i izvornom obliku. Osnovana je i tvrtka MySQL AB kako bi se omogućila distribucija MySQL-a u okviru licence otvorenog koda i komercijalnih licenci te ponudili tehnička podrška, usluge nadgledanja i obuka. 2008. godine Sun Microsystems je preuzeo MySQL AB.

U početku je MySQL postao popularan zbog svoje brzine i jednostavnosti. No, bilo je i kritika, zbog nedostatka značajki kao što su transakcije i podrška za strane ključeve. MySQL se tako nastavio razvijati dodajući i druge značajke, što je dovelo do rasta uporabe ovog sustava u području poslovnih aplikacija [1].

2.2 Osnovne značajke

U nastavku navodimo neke osnovne značajke i prednosti MySQL u usporedbi s drugim sustavima za upravljanje bazom podataka.

Brzina. Glavni cilj programera koji su implementirali MySQL bio je da uz sve zadane uvjete bude brz, tako da je stvaran s brzinom na umu.

Jednostavnost korištenja. MySQL je visoko učinkovit, ali relativno jednostavan sustav upravljanja bazom, manje složen za postavljanje i održavanje od većih sustava. Moguće je izgraditi cijelu MySQL bazu i komunicirati s njom koristeći nekoliko izjava iz SQL jezika.

Upotreba SQL-a. MySQL koristi SQL, standardizirani jezik svih modernih baza podataka.

Više mogućosti povezivanja s bazom. MySQL poslužitelj omogućuje istovremenu vezu više klijenata gdje svaki klijent može istovremeno koristiti više baza podataka. Dostupna aplikacijska i programska sučelja opisana su u idućem odjeljku.

Umreženost i sigurnost. MySQL je u potpunosti umrežen i bazi podataka se može pristupiti s bilo kojeg mjesta na Internetu što omogućuje dijeljenje podataka s drugima bez obzira na lokaciju. No, MySQL ima kontrolu pristupa tako da korisnik koji

ne bi smio vidjeti neke podatke to ni ne može učiniti. Da bi se osigurala dodatna sigurnost, MySQL podržava šifrirane veze pomoću protokola *Secure Sockets Layer - SSL*.

Prenosivost. MySQL radi na mnogim vrstama Unix baziranih sustava, ali i na drugim sustavima kao što su Windows, Mac OS i NetWare. S hardverske strane, pokriven je cijeli raspon uređaja od servera koji se upotrebljavaju u poslovnim okruženjima do osobnih računala.

Mala veličina. Skromna veličina distribucije, osobito u odnosu na neke druge "glomaznije", ostavlja dosta prostora na disku.

Dostupnost i trošak. MySQL je projekt otvorenog koda dostupan pod višestrukim licencama. Dostupan je pod uvjetima GNU Opće javne licence (*GNU General Public License - GPL*) - bez troškova za većinu upotreba unutar poduzeća. Za organizacije koje preferiraju formalne ugovore (ili ih moraju imati) ili koje ne žele biti vezane uvjetima GNU Opće javne licence, dostupne su komercijalne licence.

Otvoreni kod. MySQL je jednostavno nabaviti putem web preglednika. Ako se želi saznati po kojem algoritmu nešto radi ili napraviti sigurnosna revizija dostupan je programski kod sustava.

Podržava velike baze podataka. MySQL rukuje bazama do 50 milijuna redaka (ili više). Zadana granica na veličinu tablice je 4 gigabajta, ali se može povećati ako operacijski sustav to može podnijeti.

2.3 Korisničko i aplikacijsko programsko sučelje

Poslužitelju se može interaktivno pristupiti koristeći nekoliko sučelja koja omogućuju unos upita i pregled rezultata: klijentski komandno-linijski i grafički programi, web preglednici i aplikacije koje podržavaju ODBC i .NET (protokoli koje je razvio Microsoft). To daje mogućnost korištenja unaprijed pripremljenog klijentskog softvera ili pisanja vlastitog softvera za prilagođene aplikacije.

Od navedenih sučelja opisujemo ono koje je, uz komandno-linijskog `mysql` klijenta, korišteno u ovom radu, a to je phpMyAdmin.

phpMyAdmin je besplatan alat otvorenog koda napisan u programskom jeziku PHP, namijenjen administriranju MySQL-a pomoću web preglednika. Među zadacima koje može izvoditi su:

- izrada, izmjena ili brisanje baze podataka, tablica, polja ili redaka

- izvršavanje SQL izraza
- upravljanje korisnicima i dozvolama
- uvoz podataka iz CSV (*Comma Separated Values*) i SQL formata i stvaranje tablica iz njih

Softver, koji je dostupan na 78 jezika, održava The phpMyAdmin Project.

Mnogi programski jezici uključuju biblioteke za pristup MySQL bazama podataka. To uključuje MySQL Connector/Net za integraciju s programom Microsoft Visual Studio (najčešće se koriste jezici kao što su C# i VB) i JDBC upravljački program za programski jezik Java. Nadalje, postoji i ODBC (*Open Database Connectivity*) sučelje pod nazivom MySQL Connector. Osim toga, dostupna su programska sučelja za mnoge jezike kao što su C, Perl, Java, PHP, Python i Ruby.

Poglavlje 3

OpenStack

3.1 Što je OpenStack

OpenStack je softver otvorenog koda za stvaranje i održavanje privatnih i javnih oblaka. Riječ je o operacijskom sustav u oblaku koji upravlja velikim bazama računalnih, skladišnih i mrežnih resursa u podatkovnom centru pomoću kontrolne ploče dostupne preko web sučelja, a koja služi administratorima za nadzor i upravljanje resursima te korisnicima za opskrbu istima.

Nastao je 2010. kao rezultat suradnje Rackspace Hostinga i NASA-e u obliku projekta otvorenog koda čiji je cilj bio pomoći organizacijama da ponude *cloud-computing* servise na standardnom hardveru, uz naglasak na jednostavnost implementacije i veliku skalabilnost. Inicijalno community izdanje čiji je izvorni kod pisan u Pythonu (što se i nastavilo kroz iduća izdanja) nazvano je Austin. Novo izdanje objavljuje se svakih 6 mjeseci, a trenutno stabilno izdanje nosi naziv Ocata. Studijski primjer iz sljedećeg poglavlja proveden je na izdanju Mitaka. Kako je riječ o projektu otvorenog koda, u nastajanju novih izdanja sudjeluje zajednica korisnika, a sve vodi OpenStack Foundation, neprofitna organizacija koja nadgleda razvoj samog OpenStacka i zajednice. OpenStack Foundation podržavaju brojne tvrtke, među kojima je i Ericsson.

Mogućnosti

Openstack omogućuje korisnicima razvoj virtualnih strojeva i drugih instanci koje obavljaju različite zadatke za upravljanje okolinom u oblaku. To olakšava horizontalno skaliranje, što znači da zadaci koji imaju koristi od istovremenog pokretanja mogu jednostavno poslužiti više ili manje korisnika u datom trenutku tako što će napraviti više istovrsnih primjeraka. Na primjer, ako određena grupa korisnika radi na istovrsnim virtualnim strojevima, prilikom stvaranja instance na zahtjev moguće je automatski stvoriti novu instancu

koja će biti spremna za novog korisnika.

U nastavku navodimo glavne mogućnosti OpenStacka.

Virtualni strojevi na zahtjev. Stvaranje i snimanje trenutnog stanja virtualnog stroja čime nastaje *image* koji može poslužiti kao *backup* ili materijal za stvaranje nove virtualnog stroja sa svim potrebnim konfiguracijama i instaliranim aplikacijama.

Mreže. Automatizirano stvaranje i uređivanje mreže te smještanje novonastalog virtualnog stroja u novu ili postojeću mrežu.

Spremanje virtualnih strojeva i proizvoljnih datoteka. Kao i u slučaju lokalno pokrenutih virtualnih strojeva, dok se ista ne terminira svi podaci su sigurni.

Višestruki projekti. *Multitenancy* označava odvojenost grupa korisnika koji su u različitim projektima (*tenantima*) kako bi se izolirao pristup računalnim resursima. Omogućeno je postavljanje kvota za različite projekte i korisnike, s tim da korisnici mogu biti povezani s više projekata.

OpenStack komponente

OpenStack se sastoji od mnogo različitih komponenata. Komponente su u biti manji projekti od kojih je OpenStack i sastavljen, a u sustavu uglavnom funkcioniraju kao servisi. Kako je OpenStack veliki projekt otvorenog koda, svatko može dodati dodatne komponente koje zadovoljavaju dodatne zahtjeve na sustav. Tako se izdanje Mitaka (travanj 2016) na kojem je rađen studijski primjer sastoji od 21 projekta, a trenutno aktualno izdanje Ocata (veljača 2017) od čak 46 projekata. Svi projekti su, također, podijeljeni u 8 kategorija. U nastavku navodimo te kategorije i glavne projekte koji su sastavni dio svih izdanja koja slijede nakon izdanja Kilo (uključivo).

1. **Računanje**

   - **Nova**. Servis za implementaciju i upravljanje velikim brojem virtualnih strojeva i drugih instanci za obradu računalnih zadataka. Skupa sa Swiftom, Nova je jedna od komponenti uključenih u originalno izdanje Austin. Komunicira s drugim servisima kao što su Keystone za autentikaciju korisnika i zahtjeva, Horizon za prikaz podataka na web sučelju i Glance za opskrbu *imageima*.

- **Glance**. Servis za otkrivanje, registraciju i preuzimanje *imagea* virtualnih strojeva. *Image* je kopija svih sadržaja nekog uređaja za pohranu (na primjer, tvrdi disk, DVD/CD), a koja uključuje sve informacije o particijama, sektorima za dizanje sustava (*boot sectors*), alokacijskoj tablici datoteka, instalaciji operacijskog sustava i aplikacijskom softveru. Glance ima aplikacijsko programsko sučelje koje omogućava ispitavanje metapodataka *imagea* kao i dohvaćanje stvarnog *imagea*.
- **Ironic**. Cilj ovog servisa je pružanje *bare metal* strojeva umjesto virtualnih strojeva. U računalnoj praksi, *bare metal* je računalo koje izvršava upute izravno na logičkom hardveru bez intervencijskog operacijskog sustava.

2. **Pohrana, sigurnosne kopije i oporavak**

   - **Swift**. Sustav za pohranu objekata i datoteka. Umjesto tradicionalne ideje o upućivanju na datoteke prema njihovoj lokaciji na disku, programeri umjesto toga mogu koristiti jedinstveni identifikator koji se odnosi na datoteku ili podatke i pustiti OpenStacku da odluči gdje pohraniti te podatke i kako najbolje osigurati sigurnosne kopije podataka u slučaju kvara stroja ili mrežne veze.
   - **Cinder**. Servis za pohranu blokova napravljen kako bi krajnjem korisniku predstavio resurse za pohranu podataka koje Nova servis može iskoristiti. Virtualizira upravljanje uređajima za pohranu blokova i pruža krajnjim korisnicima aplikacijsko programsko sučelje za slanje zahtjeva i potrošnju tih resursa bez potrebe za poznavanjem gdje je njihova pohrana zapravo implementirana ili na koju vrstu uređaja.

3. **Mreže i dostava sadržaja**

   - **Neutron**. Servis koji pruža "mrežnu povezanost kao uslugu" (*"network connectivity as a service"*) između sučelja kojima upravljaju drugi servisi (npr. Nova) .

4. **Podaci i analiza**

   - **Sahara**. Programski okvir (*framework*) za *big data* procesiranje; pruža jednostavan način pokretanja aplikacijskog klastera (Hadoop ili Spark) na OpenStacku.

5. **Sigurnost, identitet i usklađenost**

   - **Keystone**. Servis za autentikaciju i autorizaciju na temelju žetona. To je u osnovi središnji popis svih korisnika OpenStack oblaka.

6. **Alati za upravljanje**

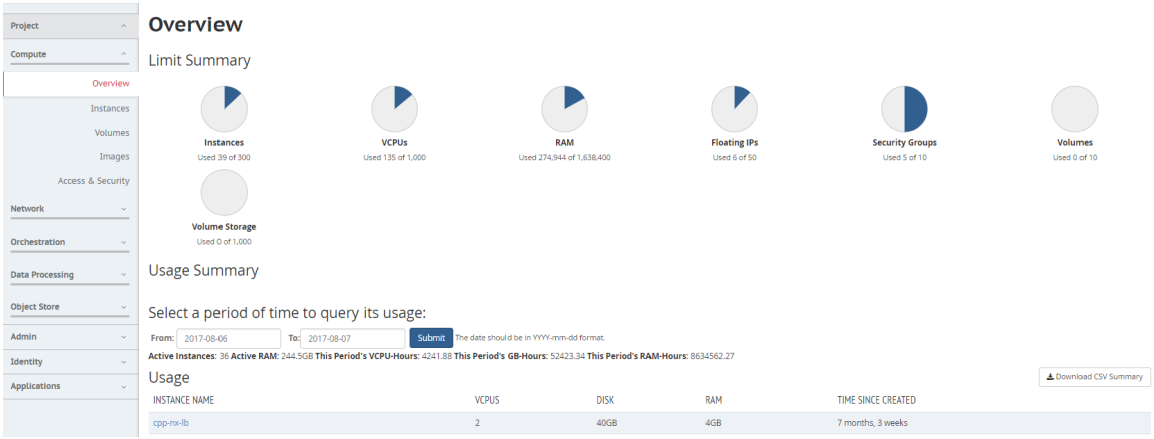
   - **Horizon**. Nadzorna ploča i jedino grafičko sučelje za OpenStack. Svim komponentama OpenStacka moguće je pristupiti putem aplikacijskog programskog sučelja, a Horizon administratorima sustava omogućuje pogled na ono što se događa u oblaku i upravljanje njime.

7. **Aplikacijski servisi**

   - **Heat**. Orkestracijska komponenta OpenStacka koja programerima omogućuje spremanje zahtjeva aplikacije u oblaku u datoteku koja definira resurse potrebne za tu aplikaciju. Na taj način se pomaže upravljanje infrastrukturom i aplikacijama unutar OpenStack oblaka.

8. **Alati za nadgledanje i mjerenje**

   - **Ceilometer**. Servis za mjerenje i prikupljanje podataka. Pretražuje mjerne podatke, događaje i poruke među servisima. Pruža telemetrijske usluge koje omogućuju naplatu pojedinačnim korisnicima oblaka, ukoliko takvi postoje.



Slika 3.1: Dio Horizon grafičkog web sučelja

Modeli korištenja OpenStacka

Modeli korištenja OpenStacka razlikuju se ovisno o potrebama organizacije, proračunu i stručnosti. U nastavku slijedi opis najčešćih modele gdje korisnik može biti pojedinac ili

tvrtka. Prve dvije mogućnosti odnose se na instalaciju u vlastititim postrojenjima, a druge uz *hosting providera*.

DIY. DIY (Do-It-Yourself) model se odnosi na preuzimanje OpenStackovog izvornog koda i implementiranje u vlastiti podatkovni centar bez vanjske pomoći. DIY je jeftin, međutim, zahtjeva znatnu *in-house* ekspertizu da bi bio izvediv izbor. S obzirom na nestašicu OpenStack inženjera može dovesti do dodatnog troška.

Distro. Korisnik preuzima i instalira OpenStack distribuciju na vlastitoj internoj mreži. U odnosu na DIY, korištenje OpenStack distribucije pruža veću stabilnost i podršku tvrtke koja tu distribuciju nudi, iako s potencijalno većim troškom uz pretpostavku da je kupljena podrška. Prema istraživanju savjetodavne tvrtke za istraživanje informacijskih tehnologija 451 Research [10] distribucija ima niži ukupni trošak vlasništva od DIY ako tvrtka može uštedjeti najmanje 45% troškova svoje radne snage upotrebom distribucije.

Upravljani privatni oblak. *Hosting provider* omogućava sve prednosti privatnog oblaka uz opciju upravljanja oblakom.

Javni oblak. Korištenjem OpenStack javnih oblaka korisnik dobiva pristup računalnim, mrežnim i skladišnim resursima uz plaćanje samo onoga što koristi.

OpenStack distribucije i Mirantis OpenStack

Na OpenStackovim službenim stranicama trenutno su dostupne 34 testirane distribucije raznih tvrtki. Distriucije se međusobno razlikuju po uključenim OpenStack servisima, podržanim hipervizorima i *guest* operacijskim sustavima te dodatnim mogućnostima implementiranima od strane tvrtke vlasnika distribucije. Hipervizor je računalni softver, *firmware* ili hardver koji stvara virtualne strojeve i upravlja njima. Računalo na kojem hipervizor pokreće jedan ili više virtualnih strojeva naziva se stroj domaćin. O hipervizorima će biti još riječi u odjeljku 4.3.

Kako su mjerenja na virtualnoj okolini iz četvrtog poglavlja (4.3) rađena na Mirantis Openstack distribuciji dalje ju pobliže opisujemo. Koraci potrebni da bi se od OpenStack izvornog koda došlo do ove distribucije su sljedeći:

Kaljenje. Popravak bugova i sigurnosnih problema.

Pakiranje. ”Pakiranje” projekata koji su spremni za upotrebu s *middlewareom* kako bi se stvorile referentne arhitekture koje dokazano rade.

Upravljanje. Stvaranje alata, metodologije i dokumentacije za početno postavljanje i kontinuirani rad kroz OpenStack Fuel, ažuriranja i nadogradnje.

Interoperabilno testiranje. Validacija *drivera* i dodataka trećih strana.

Fuel, uz Murano, Saharu i Rally, spada u projekte koje vodi upravo Mirantis i velika je pomoć u procesu instalacije i upravljanja opremom u OpenStack okolinama. Mirantis OpenStack je najfleksibilnija distribucija OpenStacka - ne ograničava korisnike u pogledu integracije s određenim komponentama kao što su hipervizori, operacijski sustavi, pohrana, softverski definirano umrežavanje (*Software-Defined Networking*), platforma kao usluga *(PaaS)* ili platforma za upravljanje oblakom (*Cloud Management Platform*). Tako Mirantis OpenStack podržava KVM (*Kernel-based Virtual Machine)*, QEMU (*Quick Emulator*) i Xen hipervizore te Windows i Linux *guest* virtualnog stroja.

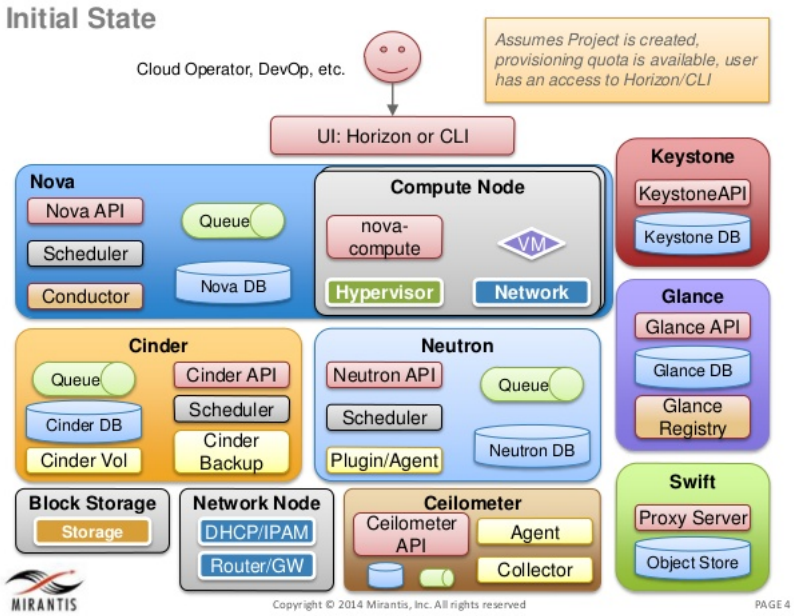
Slijedi popis OpenStack servisa uključenih u Mirantis OpenStack gdje se izdanje odnosi na OpenStack izdanje.

Servis	**Izdanje**
Application Catalog API	Mitaka (Murano v1.0)
Bare Metal Provisioning Service	Mitaka (Ironic)
Block Storage API & Extensions	Mitaka (Cinder v2.0)
Compute Service API & Extensions	Mitaka (Nova v2.1)
Dashboard	Mitaka (Horizon)
Big Data Processing Framework Provisioning API	Liberty (Sahara 1.1)
Identity service API & Extensions	Mitaka (Keystone v3.0)
Image Service API	Mitaka (Glance v2.2)
Networking API & Extensions	Mitaka (Neutron v2.0)
Orchestration API	Mitaka (Heat v1.0)
Metering & Data Collection Service API	Mitaka (Ceilometer v2.0)

Primjer: stvaranje virtualnog stroja u OpenStacku

Korisnik se prijavljuje putem korisničkog sučelja koje može biti komandna linija ili Horizon. U ovom primjeru pretpostavljamo da riječ o Horizonu i da već postoji neki OpenStack projekt, tj. okolina. Nastale virtualne strojeve nazivamo i instancama. Na slici 3.2 vidljive su sve glavne OpenStack komponete u početnom stanju.

1. Korisnik specificira parametre virtualnog stroja (nove instance): ime, *flavor* (broj procesora, veličinu radne memorije i diska), mrežu u kojoj će se instanca nalaziti, opcionalno SSH ključeve za spajanje na instancu i drugo. Parametri forme se šalju kao HTTP POST zahtjev:
    - Keystone servisu ako žeton nije u priručnoj (*cache*) memoriji. Slijedi *korak 2*.
    - Aplikacijskom programskom sučelju servisa Nova (Nova API) ako žeton još nije istekao. Slijedi *korak 3*.



Slika 3.2: Početno stanje u procesu kreiranja virtualnog stroja

2. Horizon šalje HTTP zahtjev Keystoneu. Sve potrebne informacije specificirane su u HTTP zaglavljima koje Keystone parsira te obavlja autentikaciju, kontrolu pristupa i autorizaciju. Ako je sve u redu, Keystone HTTP-om vraća Horizonu privremeni žeton koji sadrži popis projekata kojima korisnik pripada i ovlaštenja koja ima u njima.

3. Horizon šalje POST zahtjev, potpisan s još važećim ili dobivenim žetonom, Nova aplikacijskom programskom sučelju (Nova API).

4. Nova API šalje dobiveni žeton Keystoneu. Keystone validira žeton i šalje HTTP odgovor s informacijama o prihvaćanju ili odbijanju istog.

5. Nova API dalje validira parametre zahtjeva. Tipografske greške se provjeravaju na razini koda, a parametri povezani s oblakom preko zahtjeva upućenih Nova bazi podataka. To je relacijska baza (većina implementacija je napravljena s MySQL-om ili PostgreSQL-om) koja čuva trenutno stanje svih objekata u računalnom klasteru. Ako zahtjev ne može biti procesiran baca se iznimka, a ako može, inicijalno stanje se sprema u bazu.

6. Nova API šalje poruku sa sljedećim akcijama i informacijama o virtualnom stroju u red čekanja poruka (Message Queue). OpenStack koristi red čekanja poruka za

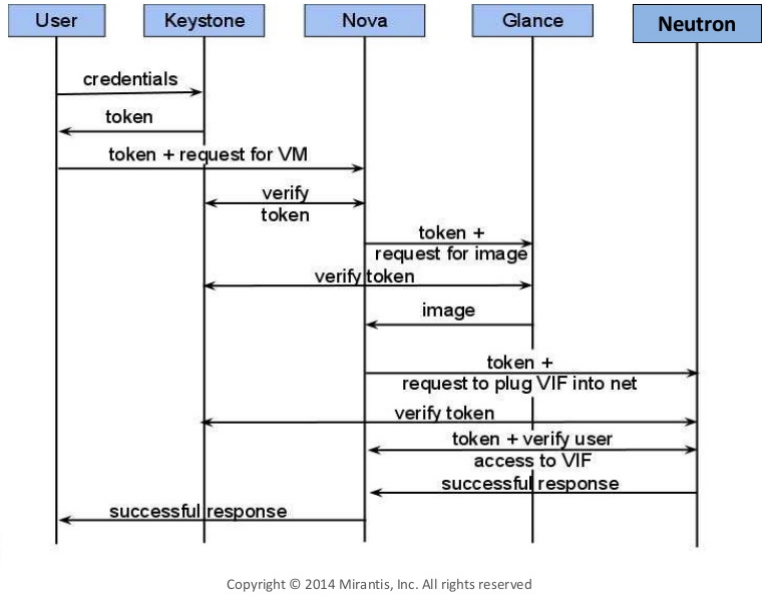
koordinaciju operacija i statusnih informacija među servisima. Podržava više servisa za red čekanja poruka, ali većina distribucija implementira RabbitMQ servis pa tako i Mirantis OpenStack.

7. Raspoređivač (Nova Scheduler) uzima poruku iz reda čekanja poruka. On odlučuje na kojem se računalnom domaćinu (*compute hostu*) zahtjev treba izvoditi pri čemu komunicira s Nova bazom podataka. Na kraju stavlja poruku s identifikatorom domaćna u red čekanja kako bi potaknuo stvaranje virtualnog stroja.
8. Računalni čvor (*compute node*) prima poruku iz reda čekanja poruka. Novom porukom iz Nova baze podataka dobiva informacije o virtualnom stroju.
9. Računalni čvor radi upit prema Neutronu radi alokacije mrežnih resursa. Neutron konfigurira IP adresu, *gateway*, DNS (*Domain Name Server* i drugo.
10. (opcionalno) Računalni čvor radi upit prema Cinderu radi alokacije logičkog diska.
11. Računalni čvor dalje radi zahtjev za *imageom* virtualnog stroja s danim identifikatorom. Ako je *image* s danim identifikatorom nađen, Glance vraća URI.
12. Računalni čvor sada pomoću URI-ja može preuzeti *image* od servisa Swift.
13. Sve informacije se (u jednoj poruci) prosljeđuju hipervizoru. Hipervizor stvara instancu. Ažurira se stanje virtualnog stroja u Nova bazi podataka.

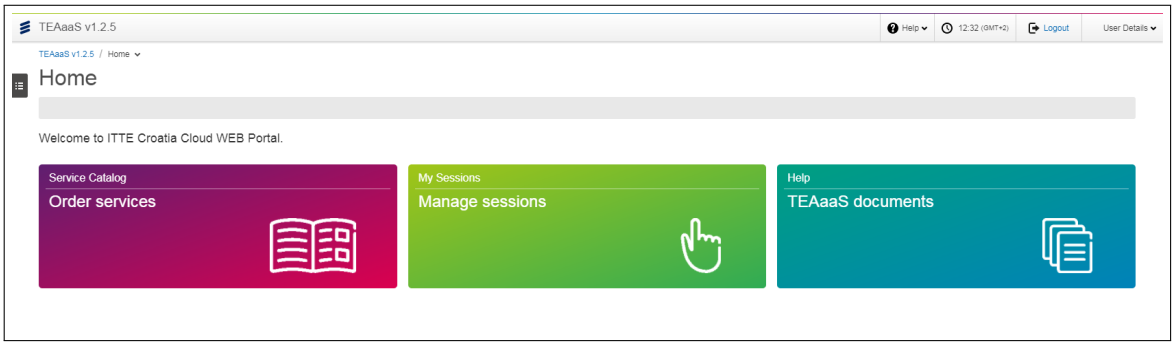
3.2 TEAaaS OpenStack okolina

Test Environment and Application as a Service (TEAaaS) je rješenje u oblaku temeljeno na OpenStacku razvijeno u ITTE odjelu tvrtke Ericsson Nikola Tesla. Cilj TEAaaS-a je pružanje informatičke tehnologije, proizvoda i usluga za razvoj i testiranje softverskih produkata. Korisnici imaju mogućnost biranja između prilagođenih aplikacija, alata i okruženja predstavljenih u kategoriziranom katalogu usluga na TEAaaS Cloud Web Portalu (CWP). CWP je web aplikacija korištena kao primarno korisničko sučelje za TEAaaS. U nastavku opisujemo tri glavne komponente CWP-a, vidljive na slici 3.4.

Katalog servisa. Sadrži popis svih trenutno dostupnih usluga u TEAaaS-u, uključujući status usluge i moguće akcije u kontekstu trenutnog korisnika. Lista autoriziranih servisa spremljena je u operacijskoj bazi podatka. Popis dostupnih servisa vidljiv je na slici 3.5. Servisi i stavke iz kataloga stvari (*Catalogue of things*) povezani su s tenantima (projektima) u OpenStacku i stvaranje novog virtualnog stroja tog servisa ili stvari znači stvaranje novog virtualnog stroja (instance) u odgovarajućem *tenantu* na način opisan u sekciji 3.1.

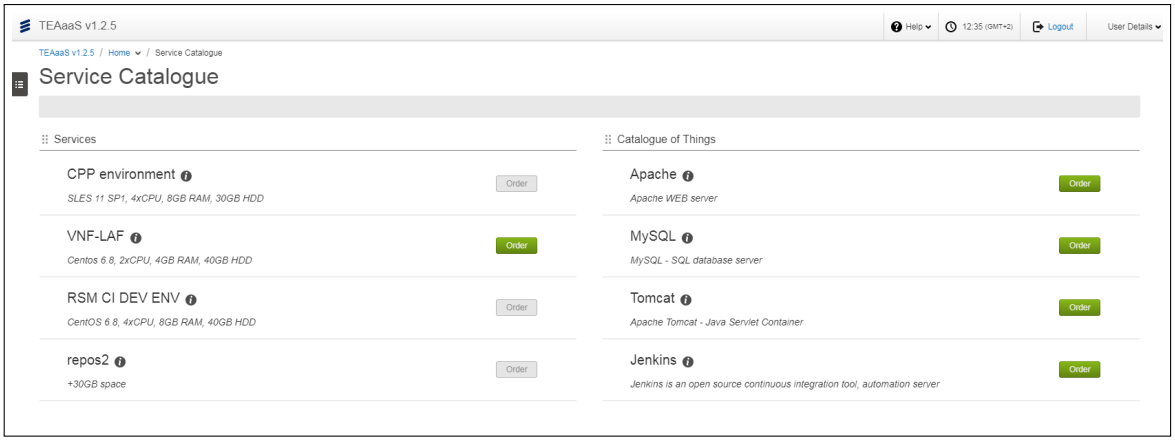


Slika 3.3: Razmjena žetona između Keystonea i ostalih servisa u procesu kreiranja virtualnog stroja



Slika 3.4: Početna stranica Cloud Web Portala prijavljenog korisnika

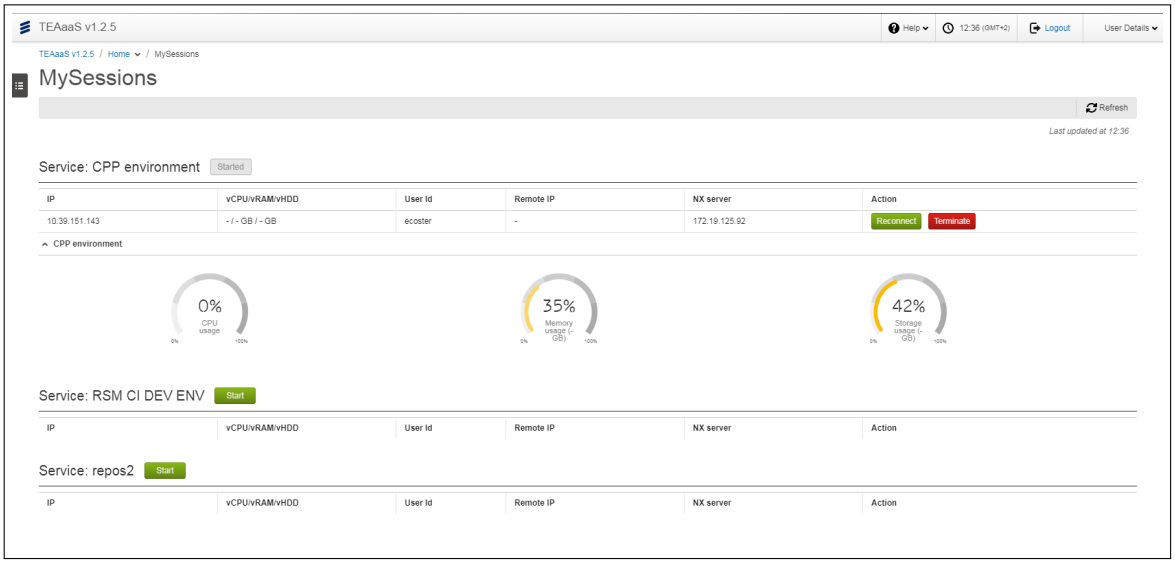
Upravljanje sesijama. Ova komponenta sadrži pristup dostupnim uslugama, prikaz trenutno korištenih resursa (CPU, memorija, prostor na disku) i informacije o prethodno kreiranim virtualnim strojevima koji su još uvijek u uporabi. Korisnik se može ponovno povezati s postojećim virtualnim strojevima (obnoviti sesiju) ili terminirati sesiju, a odabirom nove usluge potrebna okolina se isporučuje putem novog virtualnog stroja. Za određenu okolinu (npr. CPP environment) inicijalno postoji predefini-



Slika 3.5: Katalog servisa dostupnih preko Cloud Web Portala

rani broj spremnih virtualnih strojeva koji čekaju na spajanje korisnika. Kako jedan korisnik "zauzme" virtualni stroj, automatski se stvara novi kako bi spreman dočekao novog korisnika i izbjeglo se čekanje koje je potrebno za konfiguraciju svih potrebnih alata i programa na svježe stvorenoj instanci. Nakon što korisnik odabere željenu aplikaciju ili okruženje, zahtjev se obrađuje i šalje OpenStacku pomoću komponente Free NX Load Balancer, nastale ažuriranjem konfiguracije Free NX Servera. Load Balancer, dakle, brine o stvaranju instanci i njihovom dodjeljivanju korisnicima. Za pokretanje sesije na računalu potrebno je imati instaliranu klijentsku komponentu - NX Client. Prikaz stranice upravljanja sesijama vidljiv je na slici 3.6.

Pomoć i podrška. Dokumentacija, podrška, linkovi za pomoć i kontakt.



Slika 3.6: Upravljanje sesijama na Cloud Web Portalu

Poglavlje 4

Studijski primjer i rezultati rada

Ovaj studijski primjer napravljen je u organizacijskoj jedinici ITTE ETK/WI tvrtke Ericsson NIkola Tesla d.d. Mjerimo zauzeće diskovnog prostora na lokalnim projektnim diskovima smještenima na fizičkim serverima i potrošnju računalnih resursa u OpenStack virtualnim okolinama unutar projekta TEAaaS. Podaci s fizičkih servera se vizualiziraju i radi se predviđanje budućih vrijednosti, a na virtualnoj okolini opisujemo i testiramo moguće načine dohvata traženih podataka koji se dalje isporučuju trećoj strani na obradu. Svi izmjereni podaci spremaju se u dvije baze podataka - big_data i teaaas, a prikupljaju ih Perl i PHP skripte koje automatski pokreće Unixov servis Cron u zadanim intervalima.

4.1 Baze podataka

Postojanje dvije različite baze koje imaju zajedničkih dodirnih točaka opravdavamo različitim administratorima i prvotnim različitim svrhama. Baza teaas je originalno služila samo za praćenje sesija virtualnih strojeva i *backend* Cloud Web Portala, dok je baza big_data napravljena sa svrhom dugoročne analize potrošnje računalnih resursa. S vremenom su se i bazi teaas neki podaci počeli trajno spremati i koriste se u svrhu analize, a podaci iz baze big_data također koriste na Cloud Web portalu. U budućnosti je planirano spajanje ove dvije baze.

Za upravljanje spomenutim bazama big_data i teaaas koristi se MySQL sustav, distribuiran kao sastavni dio serverskih Linux distribucija. U našem slučaju je pokrenut na CentOS-u 6.5; serveru koji za potrebe ovog rada možemo nazvati guslica. Kako je riječ o verziji MySQL 5.1.71, zadani mehanizam za pohranu je MyISAM.

Baza podataka big_data

Baza podataka big_data sastoji se od 8 tablica čiju strukturu i kratki opis navodimo u nastavku. Tablice 1 - 4 se odnose na projektne diskove na fizičkim serverima, a 5 - 9 na OpenStack okolinu.

1. **Tablica particije**. Mjereni projekni diskovi s pripadajućim serverima i pridruženim identifikatorom tipa int koji se koristi u tablici ocitavanja 4.2.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
ime_particije	varchar(20)	NO		NULL	
server	int(11)	NO		NULL	

Tablica 4.1: Tablica particije

2. **Tablica ocitavanja**. Podaci o ukupnoj veličini, iskorištenom i slobodnom prostoru na projektnim diskovima iz tablice particije 4.1. Vrijednosti size, used i free su u gigabajtima.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
ime_particije	varchar(15)	YES		NULL	
timestamp	timestamp	NO		CURRENT_TIMESTAMP	
size	float	YES		NULL	
used	float	YES		NULL	
free	float	YES		NULL	

Tablica 4.2: Tablica ocitavanja

3. **Tablica users**. Korisnici na projektnom disku `homes`, raspoređeni u 10 direktorija - /home1, /home2, ..., /home10. Svakog korisnika određuje njegov user_id, koji je rijec nastala kombinacijom imena i prezimena. Pridružuje mu se identifikator tipa `int` i identifikator pripadnog "manjeg" home direktorija tipa `int`.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
user_id	varchar(20)	YES		NULL	
home_id	int(11)	NO		NULL	

Tablica 4.3: Tablica users

4. **Tablica homes_all**. Iz ove tablice se dobivaju podaci za TEAaaS Cloud Web Portal gdje je korisniku omogućeno da vidi koliko osobno zauzima prostora u svom `home` direktoriju. Neki korisnici (uglavnom administratori) imaju svoje direktorije u više `home` direktorija koji se svi prate. Kako bi se na Cloud Web portalu prikazao samo primarni `home` direktorij uz mjerenje se sprema i home_id, iako bi se on za većinu korisnika mogao dobiti iz tablice users. Za neke korisnike ne vrijedi da je ukupna veličina njima raspoređenog diskovnog prostra jednaka zbroju zauzetog i slobodnog prostora. Opet je uglavnom riječ o administratorima koji nemaju određenu gornju kvotu, što predstavlja dodatnu odgovornost za njih i ova mjerenja im olakšavaju uvid u stanje stvari. Vrijednosti size, used i free kolumni su u megabajtima.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	int(11)	NO	PRI	NULL	auto_increment
user_id	varchar(20)	YES		NULL	
home_id	int(11)	NO		NULL	
timestamp	timestamp	NO		CURRENT_TIMESTAMP	
size	float	YES		NULL	
used	float	YES		NULL	
free	float	YES		NULL	

Tablica 4.4: Tablica homes_all

5. **Tablica tenants**. Sadrži projekte OpenStack okoline koji se prate, dakle, za sada su tu samo CPP_tenant i RSM_CI_DEV_ENV. tenant_id se odnosi na identifikator projekta u OpenStack okolini. Određuje ga servis Nova.

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	int(11)	NO	PRI	NULL	auto_increment
tenant_name	varchar(50)	NO		NULL	
tenant_id	int(50)	NO		NULL	

Tablica 4.5: Tablica tenants

6. **Tablica users_openstack**. U ovoj su tablici svi korisnici koji se ne nalaze u tablici users, a korisnici su OpenStacka, tj. TEAaaS-a. Njihov id je četveroznamenkast i počinje s 3 kako bi se razlikovali od korisnika iz tablice users čiji id-evi idu do 1800.

7. **Tablica instances**. Svaka mjerena instanca se sprema ovdje. Kako svaka nova instanca u oba projekta tenanta (CPP_tenant i RSM_CI_DEV_ENV) koja pratimo u OpenStack okolini dobiva jedinstveni identifikator i ukupni broj instanci u ovoj tablici je veći od ukupnog broja instanci u projektima.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
user_id	varchar(1)	NO		NULL	

Tablica 4.6: Tablica users_openstack

Field	Type	Null	Key	Default	Extra
id	int(20)	NO	PRI	NULL	auto_increment
instance_id	varchar(50)	NO		NULL	
instance_addr	varchar(15)	YES		NULL	
tenant_id	varchar(50)	NO		NULL	

Tablica 4.7: Tablica instances

8. **Tablica metrics**. Popis mjerenja odabranih između onih koje nudi Ceilometer (OpenStack telemetrijski servis) s pripadajućim mjernim jedinicama.

Field	Type	Null	Key	Default	Extra
id	int(20)	NO	PRI	NULL	auto_increment
meter_id	varchar(50)	NO		NULL	
unit	int(10)	NO		NULL	

Tablica 4.8: Tablica metrics

9. **Tablica openstack_measuring**. ”Master” tablica za mjerenja u OpenStack okolini. Vrijednosti za instance_id, tenant_id, meter_id i user se prilikom unosa dohvaćaju iz tablica instances (4.7), tenants (4.5), metrics (4.8) i openstack_users (4.6).

Field	Type	Null	Key	Default	Extra
id	int(20)	NO	PRI	NULL	auto_increment
instance_id	varchar(50)	NO		NULL	
tenant_id	varchar(50)	NO		NULL	
meter_id	int(20)	NO		NULL	
value	float	NO		NULL	
timestamp	timestamp	NO		CURRENT_TIMESTAMP	
user	int(11)	YES		NULL	

Tablica 4.9: Tablica openstack_measuring

Baza podataka teaaas

Kako je riječ o nasljeđenoj bazi, navodimo samo nove tablice relevantne za ovaj rad - repos2users i measuring_ repos2. Riječ je o grupi korisnika koja je vezana za određene aktivnosti i potrebe u smislu potrebnih aplikacija, alata i računalnih resursa za implementaciju i testiranje programskih rješenja koje razvijaju, a svi koriste direktorij `repos2`.

1. **Tablica repos2users**. Korisnici iz posebne grupe koji koriste `repos2` direktorij.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
userId	varchar(15)	NO	UNI	NULL	

Tablica 4.10: Tablica repos2users

2. **Tablica measuring_repos2**. Za svakog korisnika se bilježi ukupno raspoređen prostor i slobodan prostor. repos2_id se odnosi na id korisnika iz tablice repos2users (4.10).

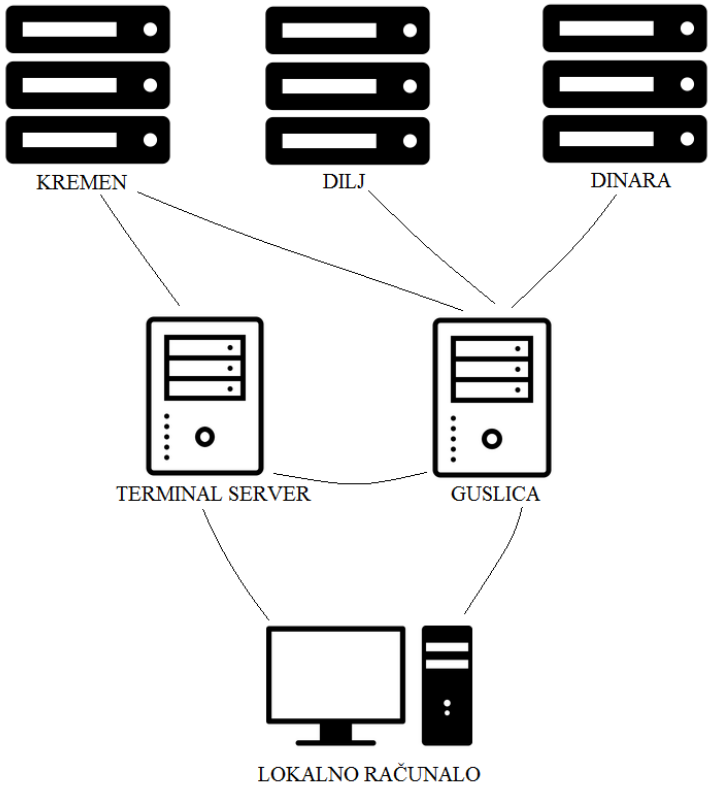
Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
repos2_id	int(11)	NO		NULL	
timestamp	timestamp	NO		CURRENT_TIMESTAMP	
available_MB	float	NO		NULL	
total_MB	float	NO		NULL	

Tablica 4.11: Tablica measuring_repos2

4.2 Mjerenja na fizičkim diskovima

Diskovi čije zauzeće promatramo nalaze se na tri servera. Za potrebe ovog rada nazvat ćemo ih dilj, dinara i kremen. Njihov razmještaj prikazuje tablica 4.12, istovjetna tablici "ocitavanja" u bazi big_data, uz proširenje stupcem u kojem je označeno je li disk *mountan* na serveru guslica, tj. možemo mu pristupiti iako pripada datotečnom sustavu drugog servera. U našem slučaju to znači da se ne moramo spajati na server na kojem se nalaze podaci koji nas zanimaju, nego ih možemo provjeriti "lokalno" na guslici gdje se i izvode sve skripte koje prikupljaju podatke.

Na slici 4.1 vidljive su veze među serverima na kojima se rade mjerenja. Za spajanje se koristi SSH veza.



Slika 4.1: Veze među serverima na kojima se rade mjerenja

Id	**ime_particije**	**Server**	**Particija *mountana* na guslici**
1	CIV	dilj	DA
2	CIV2	kremen	DA
3	CIV3	kremen	DA
4	CIV4	dinara	DA
5	CIV5	dinara	DA
6	homes	kremen	NE
7	proj	dilj	NE
8	pool3/rsm	dilj	NE
9	RSM	dilj	NE
10	proj/cpprepos	dilj	NE

Tablica 4.12: Popis mjerenih projektnih diskova

Projektni diskovi

Podaci o zauzeću projektnih diskova iz tablice 4.12 prikupljaju se svakih osam sati i spremaju u tablicu ocitavanja u bazi big_data. Diskove koji su *mountani* na serveru guslica provjeravamo na samoj guslici, dok se za ostale diskove spajamo na pripadne servere.

U nastavku slijedi programski kod Perl skripte koja prikuplja i sprema tražene podatke.

```
#!/usr//bin/perl

use DBI;

# pomocne varijable

my $output;
my $total;
my $used;
my $available;

## SQL query
$query = "SELECT * FROM particije";

$dbh = DBI->connect("DBI:mysql:$db:$host", $user, $pass);
$sqlQuery = $dbh->prepare($query)
or die "Can't prepare $query: $dbh->errstr\n";

$rv = $sqlQuery->execute
or die "can't execute the query: $sqlQuery->errstr";

my @row;
my $output;

while (@row = $sqlQuery->fetchrow_array() )
{
  my $id = $row[0]; print "Id je :$id\n";
  my $particija = $row[1];
  my $thing_that_is_measured ="/$particija";
  print "\tTrenutno mjerimo $thing_that_is_measured.\n";

    # homes
   if ( $id == 6 ) {
```

```
  $output = `ssh etk\@kremen "df -k $thing_that_is_measured" `;
  goto CONTINUE;

}
 # pool3/rsm, /RSM, proj/cpprepos
elsif ( $id > 6 ) {
  $output = `ssh etk\@dilj "df -k $thing_that_is_measured" `;
  goto CONTINUE;
}

else {
  $output = `df -k $thing_that_is_measured`;
}

if(!($output )){
    die "Error trying to get disk usage for partition
        $thing_that_is_measured\n";
    exit(1);
}

CONTINUE:
my @output_lines = split /\n/, $output;
my @values = split /\s+/, $output_lines[1];


if ( $id == 7 ){
  my $quota_output = `ssh etk\@dilj "/usr/sbin/zfs get quota proj | tail
      -1"`;
  print "\n$quota_output\n";
 my @quota_output_tmp = split /\s+/, $quota_output;
  $total = $quota_output_tmp[2];
  # u total i dalje ostaje broj, a u meter_total treba biti G ili T
 my $meter_total = chop $total;
  if ( $meter_total eq "T"){
    $total = $total * 1024;
  }
}
else{
    $total = $values[1]/1024/1024;
}

# pretvaramo u GB
```

```
  print "\ttotal: $total\n";
  $used = $values[2]/1024/1024; print "\tused: $used\n";
  $available = $values[3]/1024/1024; print "\tavailable: $available\n\n\n";

  # upis u tablicu ocitavanja
  my $sth = $dbh->prepare("INSERT INTO ocitavanja (ID, IME_PARTICIJE,
      SIZE, USED, FREE ) values (?,?,?,?,?)");
  $sth->execute('NULL', $particija, $total, $used, $available)
          or die $DBI::errstr;
  $sth->finish();

}

my $rc = $sqlQuery->finish;

$rc = $dbh->disconnect or warn $dbh->errstr;

exit(0);
```

Korisnički direktoriji

Na sličan način kao i za velike projektne diskove prate se korisnički direktoriji na projektnom disku homes. Da bismo došli do njih, spajamo se na kremen. U nastavku slijedi Perl skripta kojom se prikupljaju i spremaju podaci o zauzeću diskovnog prostora.

```
#!/usr//bin/perl

use DBI;

my $output;
my $home;
my $user;
my $output;

$dbh = DBI -> connect("DBI:mysql:$db:$host", $db_user, $pass);

my @homes = `ssh etk\@server3 ls /homes`;

# u polju @homes se sada nalaze home1, home2, ..., home10
foreach my $home(@homes) {
```

```
    chomp($home);

    # u tablici homes svakom elementu polja@ homes pridruzen identifikator#
        kojeg dohvacamo
    my $sth = $dbh -> prepare("SELECT id FROM homes WHERE home = ?");

    $sth->execute($home);
    my @row = $sth -> fetchrow_array;
    my $home_id = $row[0];

    # u polju @users su sada svi korisnici na npr.home5 direktoriju
    my @users = `ssh etk\@server3 ls /homes/$home`;

    foreach $user(@users) {
      chomp($user);
      my $output = `ssh etk\@server3 df -k /homes/$home/$user`;

      if (!($output)) {
      # ako naredba ne prodje, rijec je direktoriju koji nije korisnicki#
          pa ga jednostavno preskacemo
        next;
      }

      my @dfkh12 = split /\n/ , $output;
      my @values = split /\s+/, $dfkh12[1];

      print "$home/$user ";

      my $TOTAL = $values[1] / 1024;
      print " size = $TOTAL";
      my $USED = $values[2] / 1024;
      print " used = $USED";
      my $FREE = $values[3] / 1024;
      print " free = $FREE\n\n";

      my $sth2 = $dbh -> prepare("INSERT INTO homes_all (id, user_id,
          home_id, size, used, free ) values (?,?,?,?,?,?)");
      $sth2 -> execute('NULL', $user, $home_id, $TOTAL, $USED, $FREE)
      or die $DBI::errstr;

    }
}
```

```
$sth -> finish();
$sth2 -> finish();

my $rc = $sth -> finish;
$rc = $dbh -> disconnect or warn $dbh -> errstr;

exit(0);
```

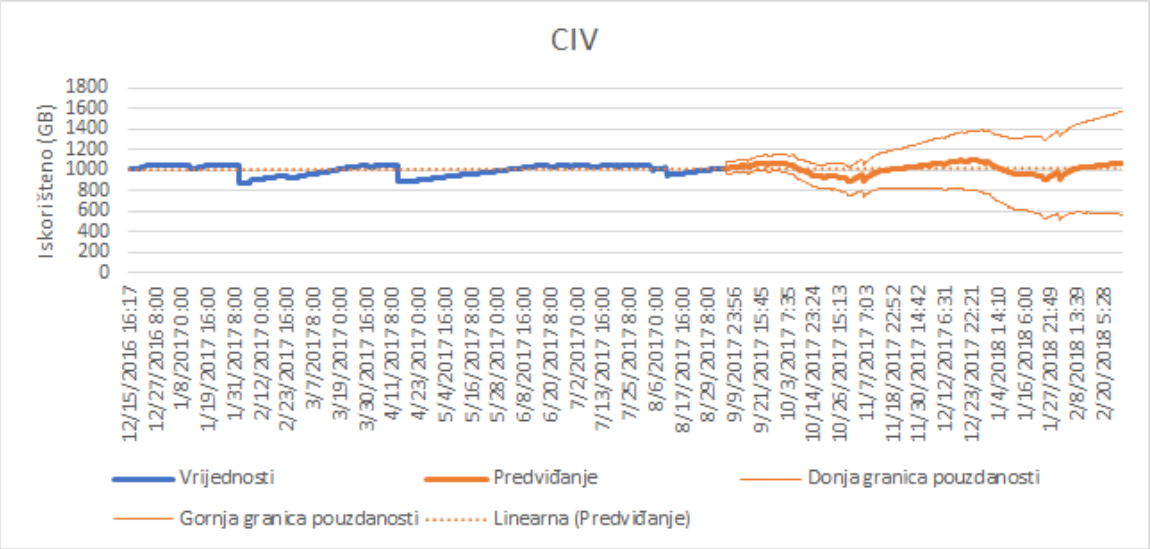
U nastavku se nalaze grafički prikazi podataka koji se prikupljaju ovom skriptom. Za projektne diskove prikazano je buduće očekivano ponašanje podataka i linearni trend rasta. Buduće vrijednosti se računaju nad postojećim podacima korištenjem algoritma eksponencijalnog zaglađivanja (*Exponential Smoothing (ETS) algorithm*). Eksponencijalno zaglađivanje [3] je tehnika "pravila palca" za "zaglađivanje" podataka vremenskih nizova. Eksponencijalne funkcije se koriste za dodjeljivanje eksponencijalno smanjujućih težina podacima tijekom vremena. Element niza podataka je obično predstavljen s x_t, počevši od trenutka $t = 0$. Izlaz algoritma eksponencijalnog zaglađivanja označen je s s_t - najbolja procjena iduće vrijednosti od x. Najjednostavnija forma eksponencijalnog zaglađivanja, počevši od trenutka $t = 0$ (uz faktor zaglađivanja $\alpha, 0 < \alpha < 1$) je

$$\begin{aligned} s_0 &= x_0 \\ s_t &= ax_t + (1-\alpha)s_t - 1, t > 0 \end{aligned} \tag{4.1}$$

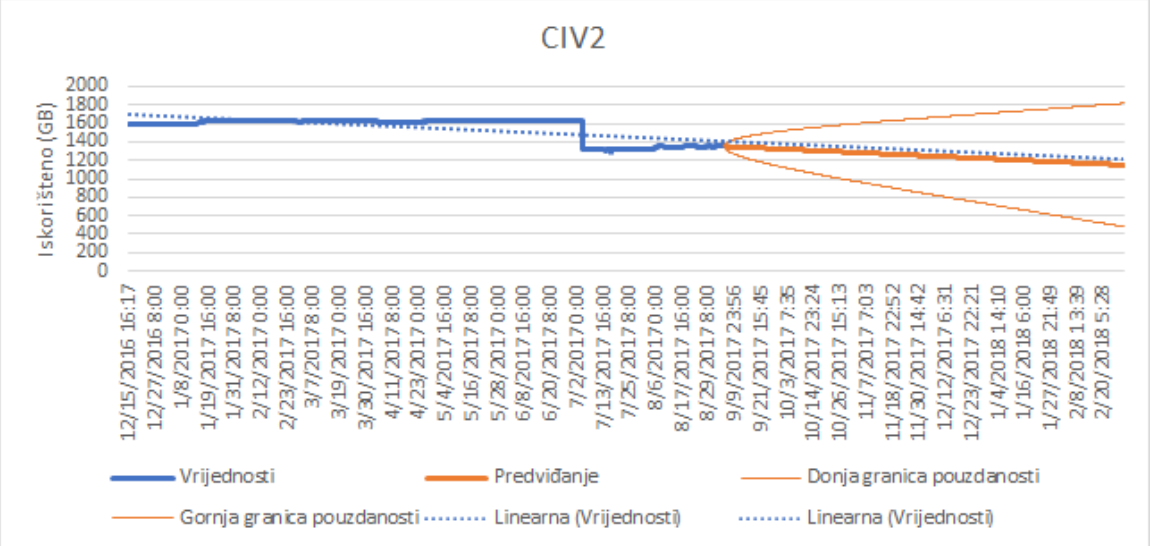
Linearni trend rasta se računa prema formuli

$$\begin{aligned} a &= y - bx \\ b &= \frac{\sum_{x,y}(x-\overline{x})(y-\overline{y})}{\sum_{x,y}(x-\overline{x})} \end{aligned} \tag{4.2}$$

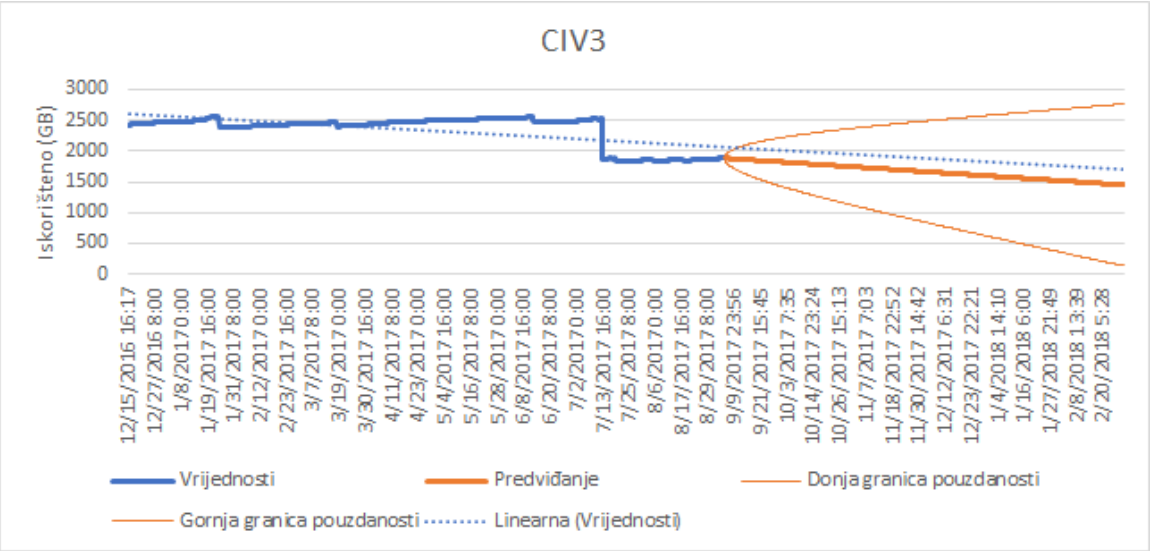
gdje su $\overline{x}$ i $\overline{y}$ srednje vrijednosti svih x, odnosno y vrijednosti iz danog skupa podataka.



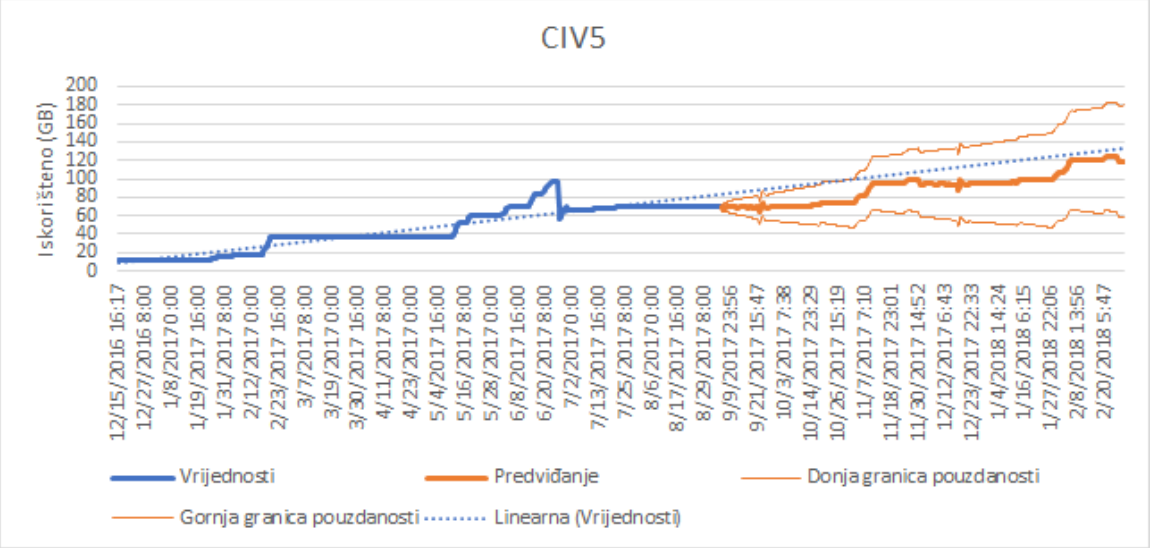
Slika 4.2: Korišteni prostor u GB kroz vrijeme na disku CIV



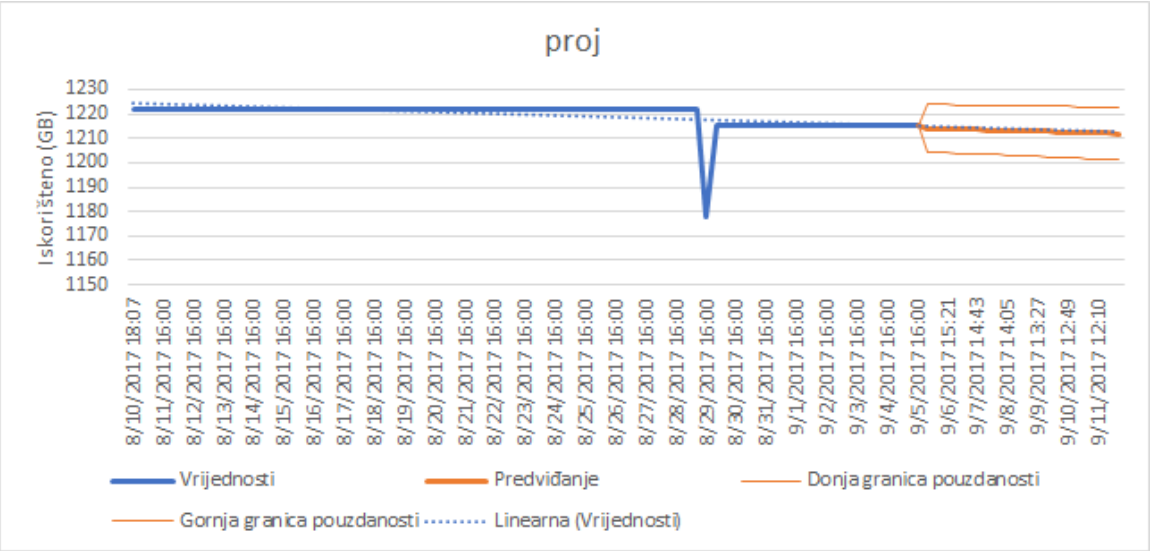
Slika 4.3: Korišteni prostor u GB kroz vrijeme na disku CIV2



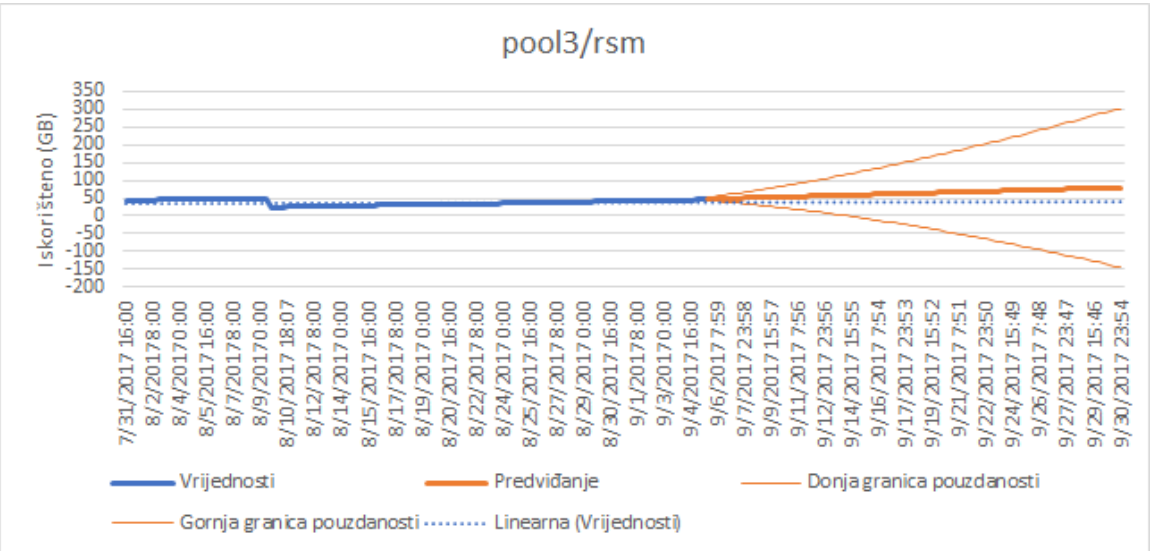
Slika 4.4: Korišteni prostor u GB kroz vrijeme na disku CIV3



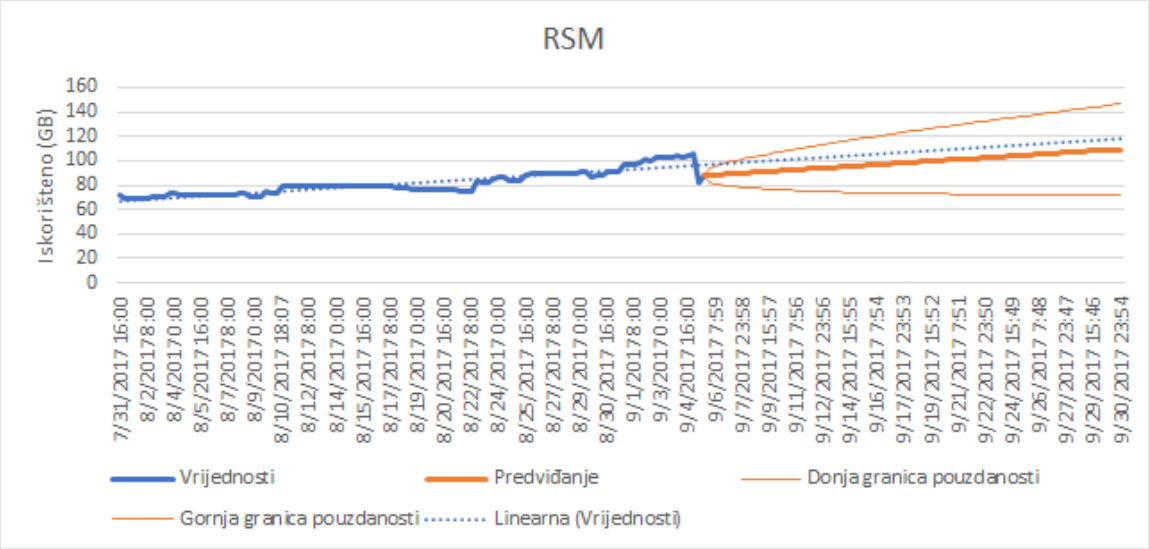
Slika 4.5: Korišteni prostor u GB kroz vrijeme na disku CIV5



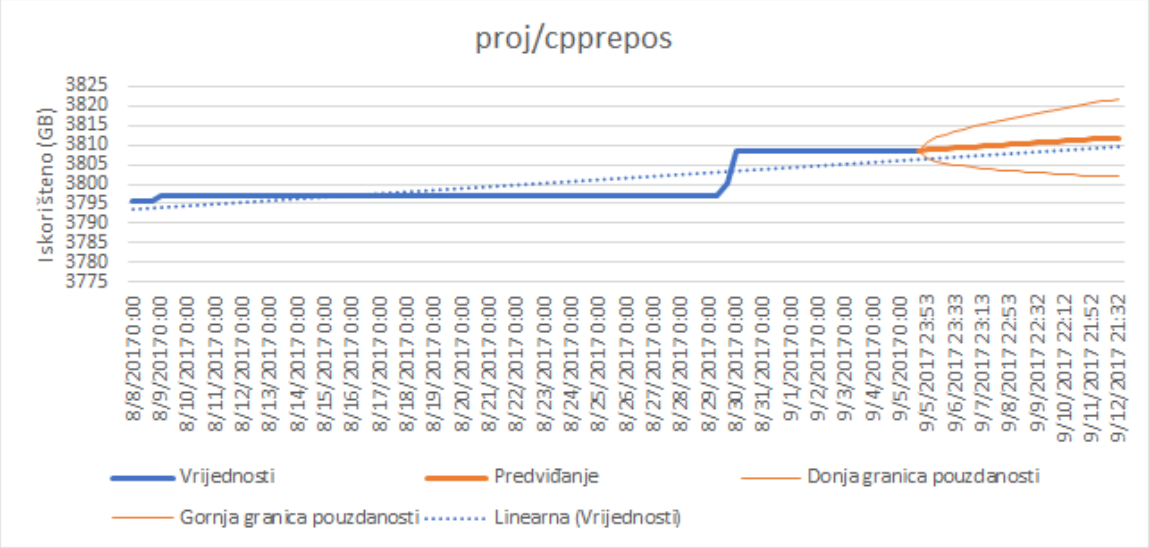
Slika 4.6: Korišteni prostor u GB kroz vrijeme na disku proj



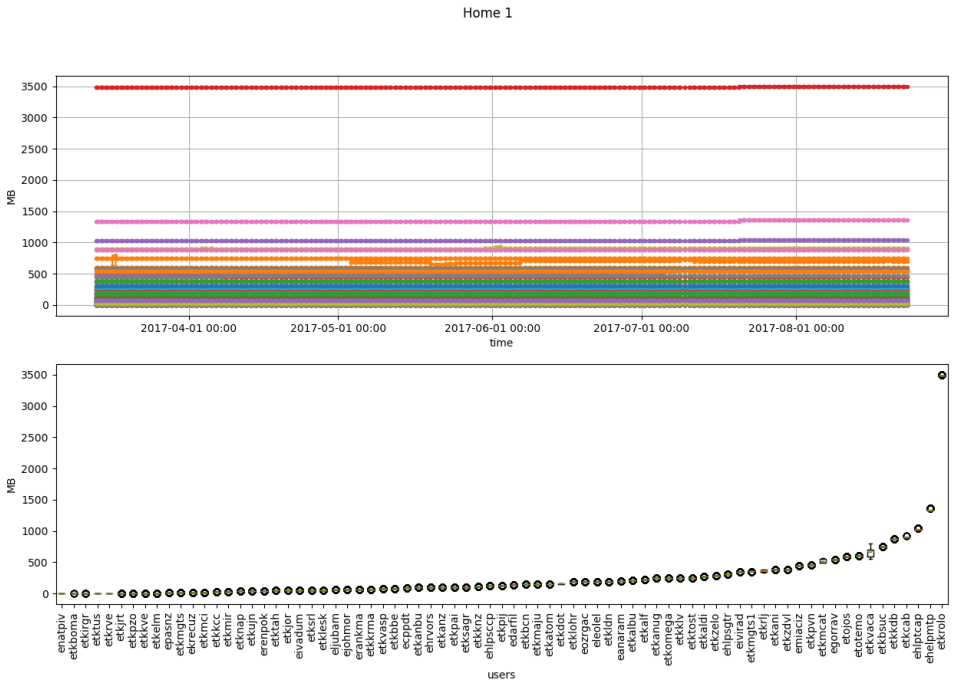
Slika 4.7: Korišteni prostor u GB kroz vrijeme na disku pool3/rsm



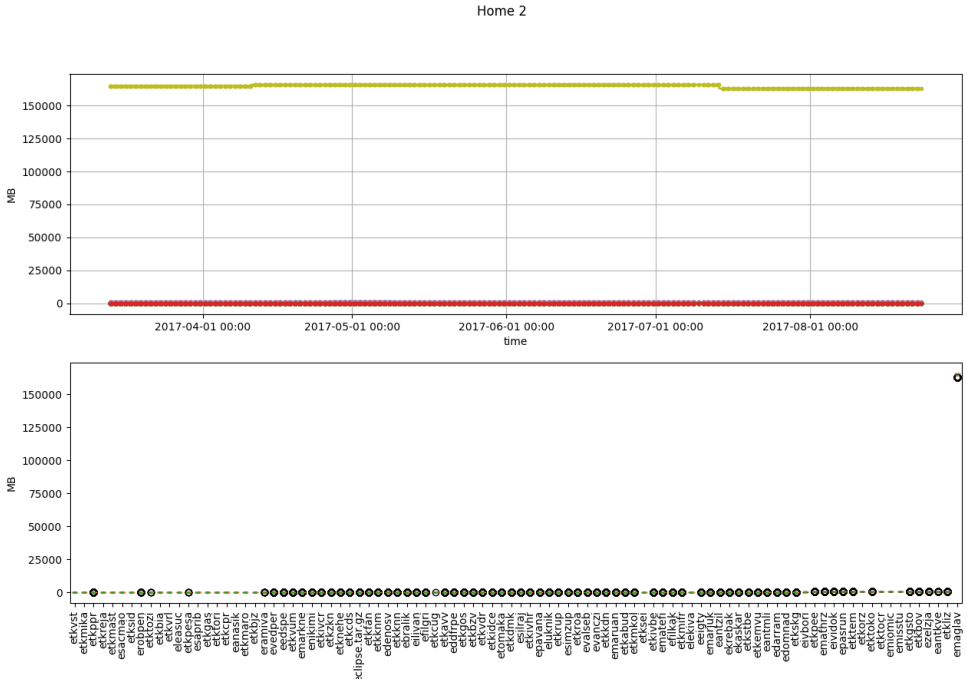
Slika 4.8: Korišteni prostor u GB kroz vrijeme na disku RSM



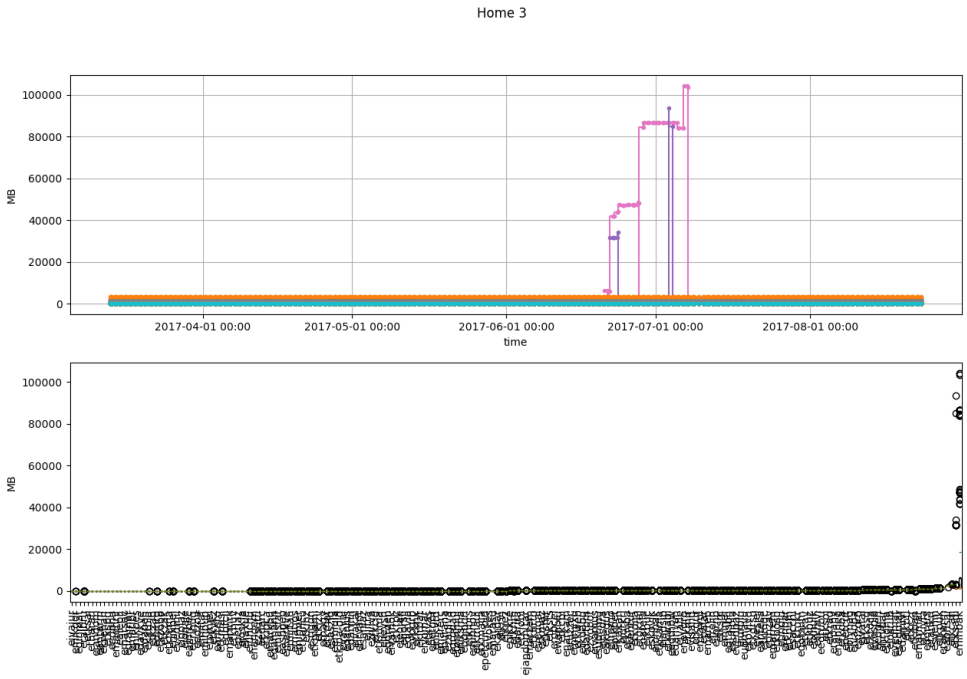
Slika 4.9: Korišteni prostor u GB kroz vrijeme na disku proj/cpprepos



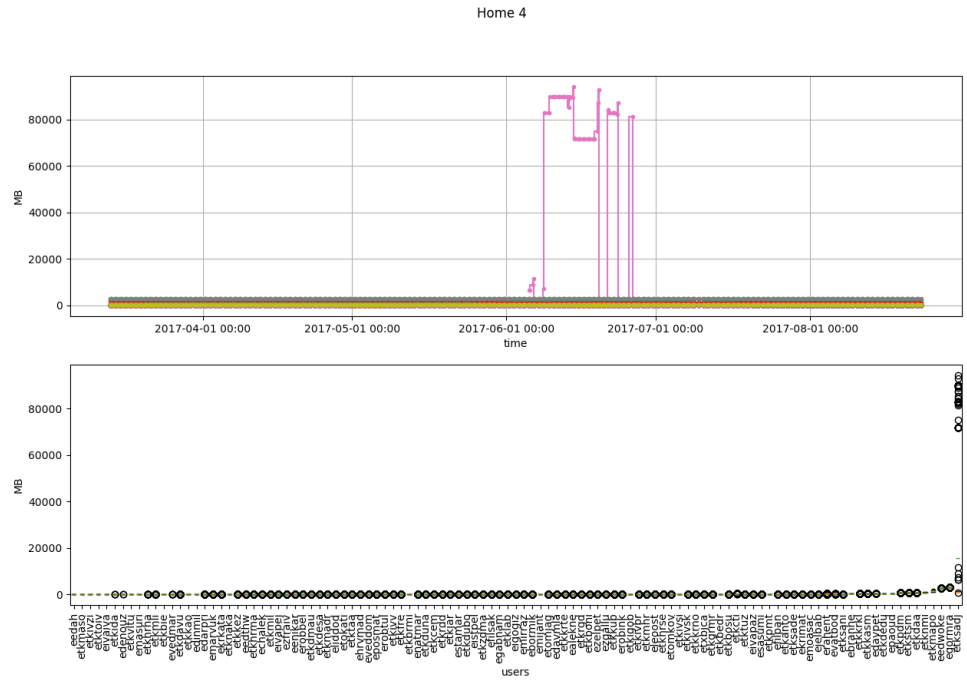
Slika 4.10: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home1



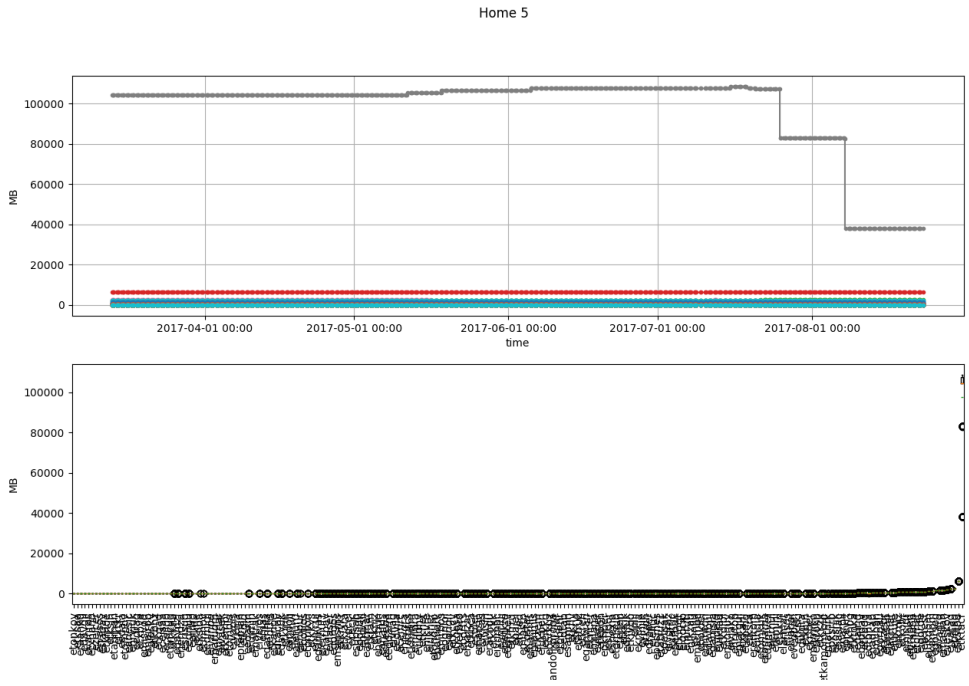
Slika 4.11: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home2



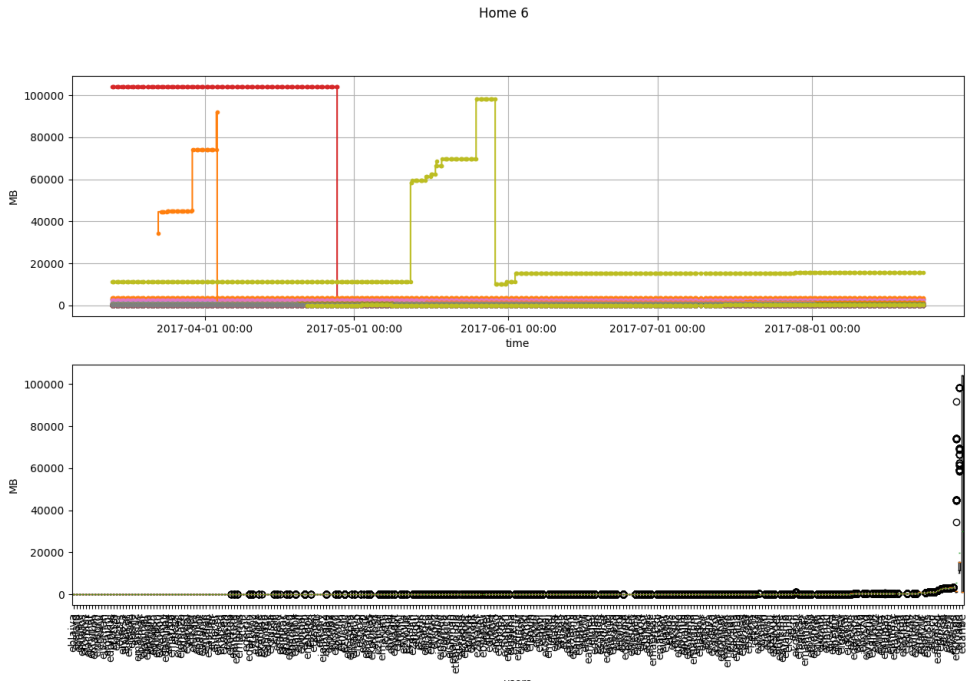
Slika 4.12: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home3



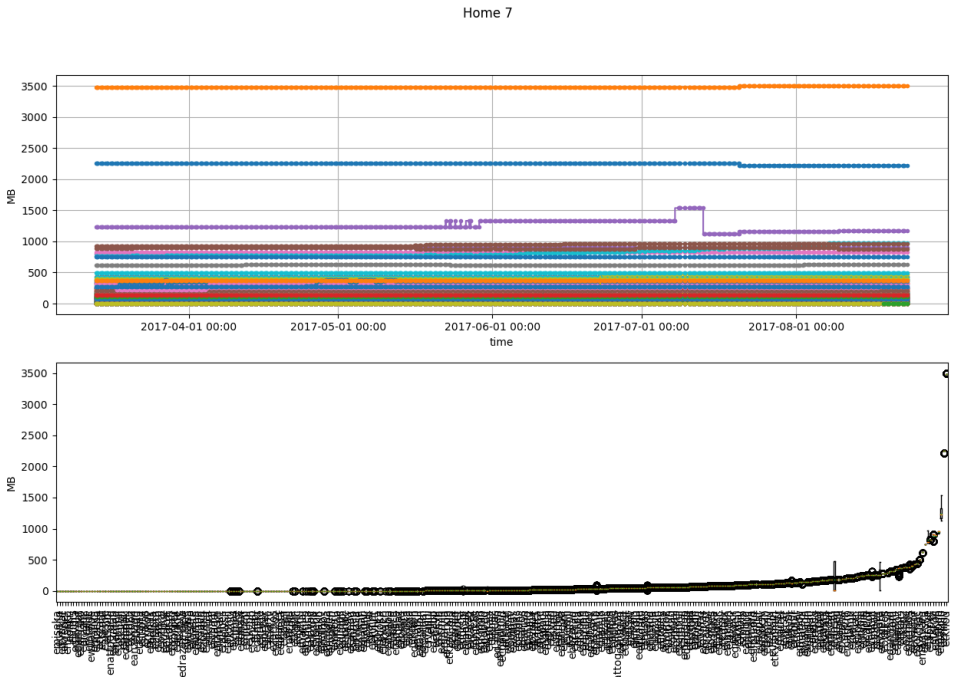
Slika 4.13: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home4



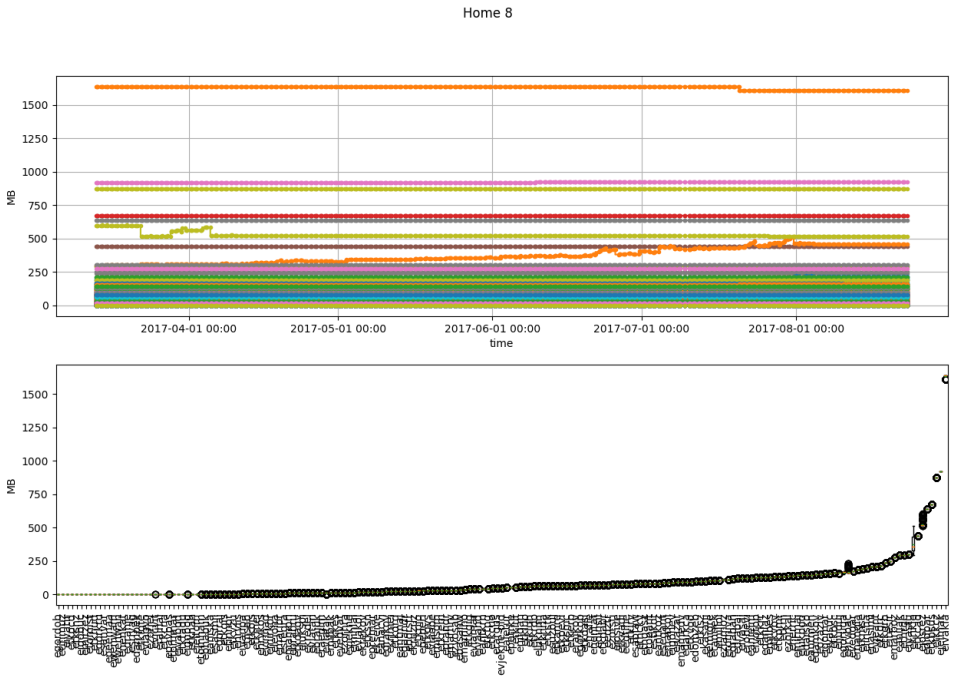
Slika 4.14: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home5



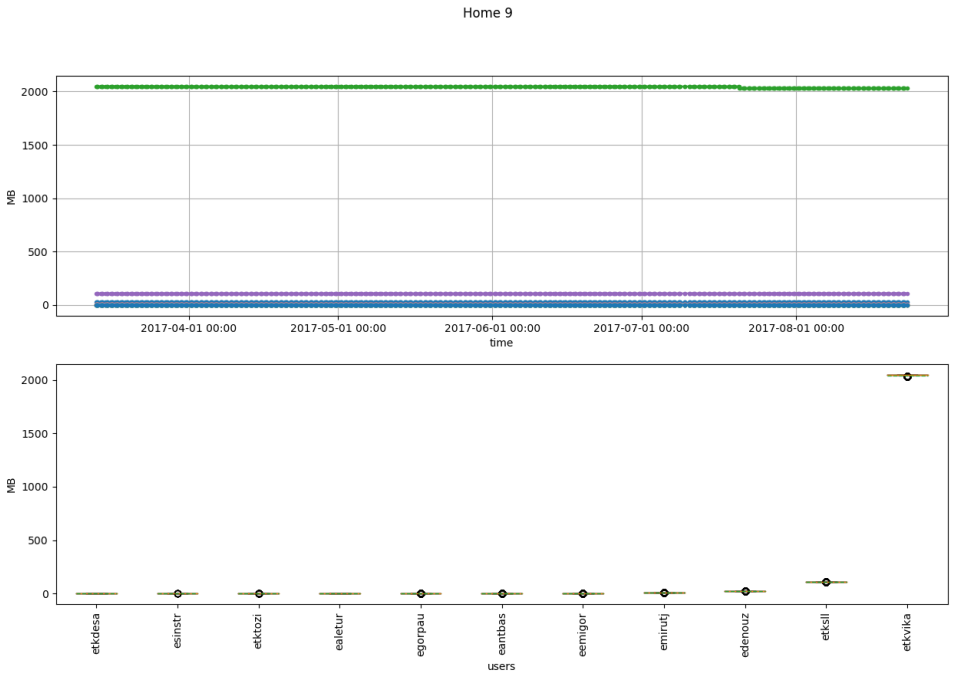
Slika 4.15: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home6



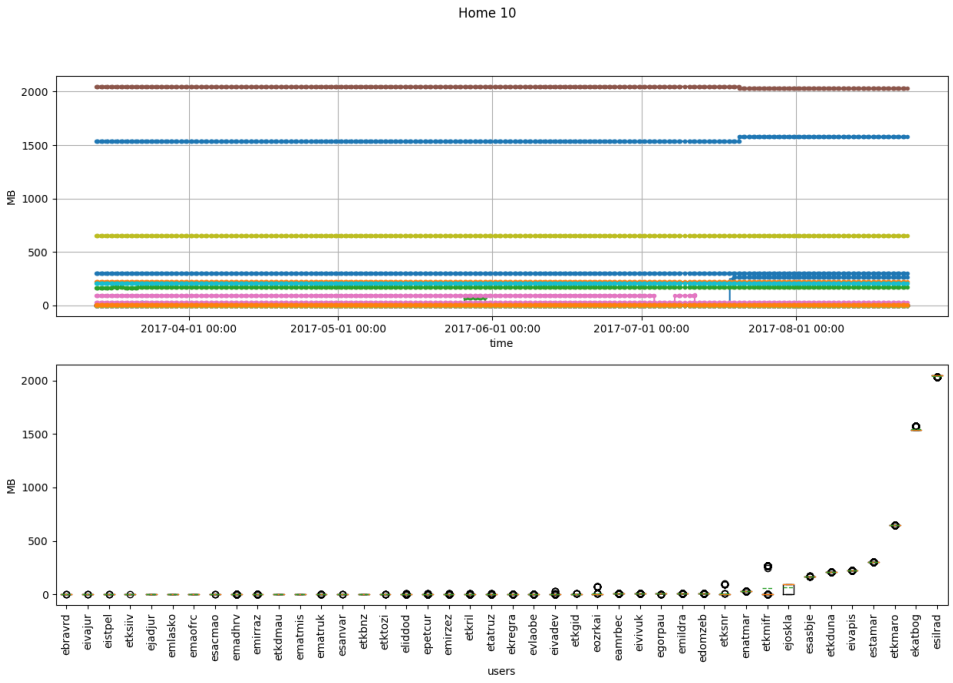
Slika 4.16: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home7



Slika 4.17: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home8



Slika 4.18: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home9



Slika 4.19: Korišteni prostor u GB kroz vrijeme i po korisnicima direktorija home10

Disk	**Linearni trend rasta**	**Procjena korištenog prostora**	**Kapacitet diska**	**Napomena**
CIV	$y = 0.0169x + 992.52$	28.2.2018. 1071.3 GB	1051.38 GB	
CIV2	$y = -0.3762x + 1704.3$	28.2.2018. 1148.88 GB	1638.4 GB	
CIV3	$y = -0.6935x + 2618.7$	28.2.2018. 1450.539 GB	2569.57 GB	
CIV4	$y = 0$	0	0	nema nacrtan graf
CIV5	$y = 0.0953x + 7.8169$	28.2.2018. 119.15 GB	97.92 GB	
homes	$y \approx 0$	≈ 0	720 GB	
proj	$y = -0.1189x + 1224.3$	12.9.2017. 1211.97	1214.99	samo mjesec dana podataka
pool3/rsm	$y = 0.0045x + 37$	30.9.2017. 79.16	100	malo više od mjesec dana podataka
RSM	$y = 0.2778x + 66.59$	30.9.2017. 109.32	300	malo više od mjesec dana podataka
proj/cpprepos	$y = 0.1484x + 3793.5$	12.9.2017. 3811.91	7280.64	samo mjesec dana podataka

Tablica 4.13: Trendovi rasta zauzeća prostora na diskovima iz tablice 4.12

Grafovi `home` direktorija predstavlju korisnike i njihovu aktivnost na način da svaka horizontalna linija gornjeg grafa predstavlja jednog korisnika (ukoliko ih više ima slične vrijednosti vizualno se spajaju u jednu liniju i razlučivi su povećanjem grafa) i promjene u vrijednostima korištenog prostora kroz vrijeme. Na donjem grafu na x-osi nalaze se korisnici (negdje jako zgusnuti jer ih je puno) i iznad njih raspone vrijednosti korištenog prostora na način da horizontalna crtica predstavlja srednju vrijednost izmjerenog zauzetog prostora, a kružić izuzetke, tako da u dosta slučajeva možemo primijetiti da veće aktivnosti pisanja ili brisanja na disku nije bi bilo pošto je izuzetak na crtici srednje vrijednosti, tj. nema ga.

U tablici 4.13 možemo vidjeti očekivane vrijednosti linearnog trenda rasta za diskove iz tablice 4.12. Kapaciteti diskova `CIV` i `CIV5` će prema očekivanim vrijednostima za pola godine biti premašeni, a kako je već u rujnu očekivana upotreba diska `proj` blizu maksimalne, puni kapacitet će vjerojatno uskoro biti dostignut.

disk	<10	<10, 30]	<30, 50]	<50, 100]	<100, 200]	>200
home1	67	4	0	2	1	1
home2	85	1	2	1	1	5
home3	196	3	4	5	4	9
home4	92	2	2	1	4	10
home5	216	3	4	3	5	9
home6	249	2	10	8	8	11
home7	305	4	4	8	4	9
home8	182	3	0	4	1	3
home9	0	1	0	0	2	8
home10	6	0	2	4	7	23

Tablica 4.14: Broj korisnika po home direktorijima i razlici minimalne i maksimalne zabilježene vrijednosti zauzeća prostora na disku

Tablica 4.14 pokazuje broj korisnika po home direktorijima i razlici minimalne i maksimalne zabilježene vrijednosti zauzeća prostora na disku. I ovdje je također vidljivo da većina korisnika ima malu aktivnost na home direktorijima.

Direktorij repos2

Ovaj direktorij posebno opisujemo jer ima i posebnu namjenu i koristi ga manja grupa korisnika u usporedbi s brojem korisnika u tablici users (4.3). Riječ je o svojevrsnom lokalnom Git repozitoriju (odatle i sam naziv `repos2`) koji služi tome da se *pull* zahtjevi ne rade u korisnikovom `home` direktoriju. Iako sami `home` direktoriji pripadaju ZFS datotečnom sustavi, oni se *mountaju* preko NFS protokola na kojem Git sporo radi, stoga se Git podaci i aktivnosti premještaju na `repos2` koji radi na `ext3` datotečnom sustavu. NFS (*Network File System*) je protokol distribuiranog datotečnog sustava koji omogućuju korisnicima na klijentskom računalu da preko računalne mreže pristupe datotekama kao da im pristupaju lokalno.

Za svakog korisnika koji se nalazi u `repos2` direktoriju želimo dobiti podatke o njemu dodijeljenom i slobodnom prostoru. Veličina samog direktorija je 3 TB i izvorno se nalazi na lokaciji `kremen:/repos`, no da bismo dobili željene podatke ne spajamo se direktno, nego preko takozvanog *terminal servera*. *Terminal server* je pristupna točka prema serverima dilj, dinara i kremen (ali ne nužno svima) u obliku Unix bazirani virtualni strojevi. Na *terminal serveru* direktorij `repos2` je *mountan* na lokaciji `/net/kremen/repos/`. Sadržaj direktorija čine stavke s korisničkim imenima. Da bismo otkrili o kakvim je datotekama riječ, u komandnoj liniji koristimo naredbu `file` za korisnika ecoster.

```
ehrzgts012:~ # file -b /net/kremen/repos/ecoster
Linux rev 1.0 ext3 filesystem data (needs journal recovery) (large files)
ehrzgts012:~ #
ehrzgts012:~ # losetup -a
/dev/loop0: [0052]:350 (/net/kremen/repos/evlafra)
/dev/loop1: [0052]:331 (/net/kremen/repos/etkboma)
/dev/loop2: [0052]:343 (/net/kremen/repos/etkvika)
/dev/loop3: [0052]:341 (/net/kremen/repos/etksadj)
/dev/loop4: [0052]:9 (/net/kremen/repos/emijant)
/dev/loop5: [0052]:344 (/net/kremen/repos/etokati)
/dev/loop6: [0052]:279 (/net/kremen/repos/ejosire)
/dev/loop7: [0052]:301 (/net/kremen/repos/emarkkn)
ehrzgts012:~ #
ehrzgts012:~ # mount | grep loop
/dev/loop0 on /repos2/evlafra type ext3 (rw)
/dev/loop1 on /repos2/etkboma type ext3 (rw)
/dev/loop2 on /repos2/etkvika type ext3 (rw)
/dev/loop3 on /repos2/etksadj type ext3 (rw)
/dev/loop4 on /repos2/emijant type ext3 (rw)
/dev/loop5 on /repos2/etokati type ext3 (rw)
/dev/loop6 on /repos2/ejosire type ext3 (rw)
/dev/loop7 on /repos2/emarkkn type ext3 (rw)
```

Slika 4.20: Provjera stavki iz direktorija repos2 i naziva *mountane* stavke

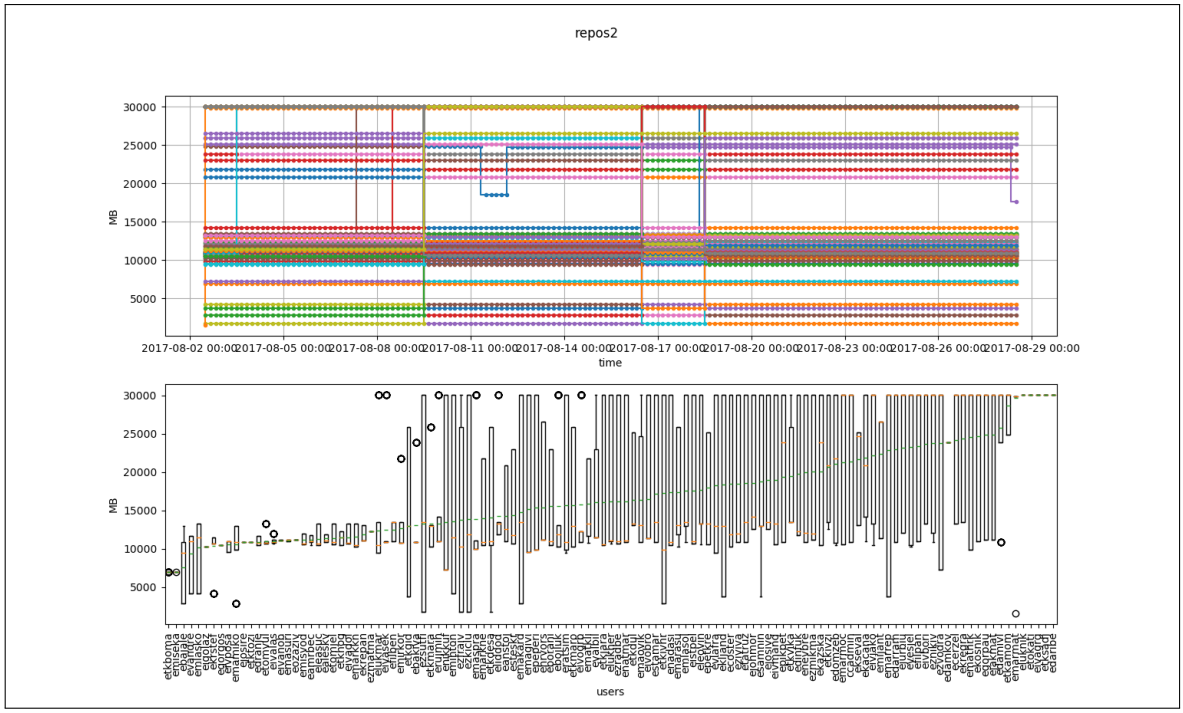
Na slici 4.20 iz odgovora `file` naredbe vidimo da je riječ o korisničkim datotečnim sustavima. Daljnjim naredbama `losetup` i `mount` objašnjava se dvojka u nazivu direktorija `repos2`- ona se odnosi na rezultat *mountanja* nekog od tih datotečnih sustava koji se potom ”ugrađuju” u `root` datotečni sustav na lokaciji `/repos2/ime_korisnika`.

U nastavku slijedi Perl skripta kojom se prikupljaju i spremaju podaci o dodjeljenom i slobodnom prostoru za svakog korisnika u direktoriju `repos2`. Podaci se spremaju u bazu teaas jer su tamo već postojale tablice vezane uz navedeni direktorij.

```
#!/usr//bin/perl

use DBI;
use Switch;
use Data::Dumper;
use POSIX;

$dbh = DBI->connect("DBI:mysql:$db:$host", $db_user, $pass);

my @repos_users = ` ssh ehrzgts030 "ls /net/kremen/repos" `;

foreach my $user (@repos_users)
{
  chomp($user);
  if ( ($user eq 'ccadmin') || ($user eq 'CPP_TCS') || ($user eq
      'vmdisks') ) {
    next;
  }

  my $row_id;

   my $sth = $dbh->prepare("SELECT id FROM repos2users WHERE userId = ?");
   $sth->execute($user);
  #$sth->finish();

  if ($sth->rows == 0) { # nema trazenog usera u tablici, treba ga dodati
    print "\tUser $user nije pronadjen u bazi, dodajem ga.\n\n";

    my $sth2 = $dbh->prepare("INSERT INTO `repos2users`(`userId`) VALUES
        (?)");
   $sth2->execute($user);
    $sth2->finish();
    $row_id = $dbh->{mysql_insertid};

    }

  else {
    @row = $sth->fetchrow_array;
    $row_id = $row[0];
  }
```

```
  my $output_total = ‘ ssh ehrzgts030 "dumpe2fs -h /net/kremen/repos/$user
      | grep 'Block count' " ‘;
  my @split_total = split /\s+/, $output_total;

  my $output_available = ‘ ssh ehrzgts030 "dumpe2fs -h
      /net/kremen/repos/$user | grep 'Free blocks' " ‘;
  my @split_available = split /\s+/, $output_available;

  my $total_MB = $split_total[2]/256;
  my $available_MB = $split_available[2]/256;
  print "\tUser: $user , total: $total_MB, available: $available_MB\n\n";

  my $sth3 = $dbh->prepare('INSERT INTO measuring_repos2 (repos2_id,
      available_MB, total_MB) VALUES    (?,?,?)');
  $sth3->execute($row_id, $available_MB, $total_MB );
  $sth3->finish();

}
```

Na slici 4.21 vidljive su veće razlike u izmjerenim vrijednostima upotrebljenog diskovnog prostora po korisniku direktorija `repos2`, što je razumljivo jer je riječ o lokalnom `git` repozitoriju koji se `pull` zahtjevima puni, a nakon finalne `push` naredbe pri završetku rada na projektu sadržaj se briše kako bi se oslobodio za nove projekte.



Slika 4.21: Provjera stavki iz direktorija repos2 i naziva *mountane* stavke

4.3 Mjerenja u virtualnoj okolini

OpenStack tenanti služe kao spremnici za resurse u oblaku koji se dodjeljuju izoliranoj grupi korisnika. U našem primjeru promatramo dva tenanta, oba uključena u projekt TE-AaaS - CPP_tenant i RSM_CI_DEV_ENV. Glavna namjena ovih *tenanta* je pružanje virtualnih strojeva prilagođenih potrebama dvije grupe korisnika. U oba *tenanta* nalazi se oko 50 instanci (virtualnih strojeva), čiji broj varira ovisno o broju korisnika i stvaranju te terminiranju instanci. Kako je opisano u odjeljku 3.2, prilikom odabira stvaranja nove instance iz odabranog *tenanta*, korisnik dobiva već unaprijed pripremljenu instancu od *tenantovog* Load Balancera. Upravo nas te, korisnički zatjevane, ”produkcijske” instance zanimaju i na njima radimo mjerenja kako bismo otkrili kako korisnici koriste računalne resurse u virtualnoj okolini.

Opis praćenih OpenStack projekata

Zajednička svrha ova dva tenanta je pružanje resursa za razvoj i testiranje softverskih rješenja, a razlikuju se u *imageima*, od kojih nastaju instance i mrežama u kojima se nalaze.

Nove instance nastaju iz snimke operacijskog sustava (skupa s instaliranim alatima). Ukoliko se ukaže potreba za novim alatom, on se instalira na jednu instancu, napravi se snimka sustava i spremi u konfiguracijske datoteke OpenStack Nova servisa i već spomenutog Load Balancera kako bi nove instance koje se budu stvarale nastale od te snimke. Korisnički zahtjevane instance (preko CWP portala) na kojima radimo mjerenja imaju SUSE Linux Enterprise Server 11 SP1 (x86_64) operacijski sustav (CPP_tenant), odnosno CentOS release 6.8 (Final) operacijski sustav (RSM_CI_DEV_ENV tenant). Imena instanci koje promatramo nose naziv "hrzgiuts" + jedinstveni slučajno određeni niz od 8 znakova, po čemu se na prvi pogled razlikuju od drugih (administratorskih, privremenih ili pomoćnih) instanci u tenantu.

Na slici 4.22 je prikazana struktura mreže u CPP_tenantu. U nastavku opisujemo općenitu strukturu OpenStack mreža i njene elemente nalazimo u prikazanoj mreži.

U OpenStack okolini svim mrežnim aspektima infrastrukture za virtualnu mrežu (*Virtual Network Infrastructure*) i aspektima pristupnog sloja infrastrukture fizičkog umrežavanja (*Physical Networking Interface*) upravlja OpenStackov servis Neutron. Neutron omogućuje tenantima stvaranje virtualnih mreža koje mogu uključivati i usluge poput vatrozida, *load balancera* i virtualne privatne mreže (VPN). Općenito, *load balancer* se koristi za distribuciju opterećenja između višestrukih *back-end* sustava ili servisa, temeljeno na kriterijima definiranima u konfiguraciji.

Svi elementi virtualne mreže, iako apstraktni objekti, oponašaju pripadni fizički uređaj. Svaki usmjernik (*router*) ima jedan pristupnik (*gateway*) koji se povezuje s mrežom i mnoga sučelja povezana s podmrežama. Podmreže mogu pristupiti instancama na drugim podmrežama povezanima s istim usmjernikem. Na slici 4.22 sve "hrzgiuts" instance su povezane unutar cpp_tenant_net mreže, a dodatno su sve i dio NFS_storage mreže koja služi za povezivanje instanci sa spreminkom podataka. Također su vidljive i druge instance unutar CPP_tenanta osim onih koje koje su praćene u ovom radu. Dvije instance koje su desno izvan mreže su u trenutku snimke bile u `error` stanju, zbog čega su tako locirane.

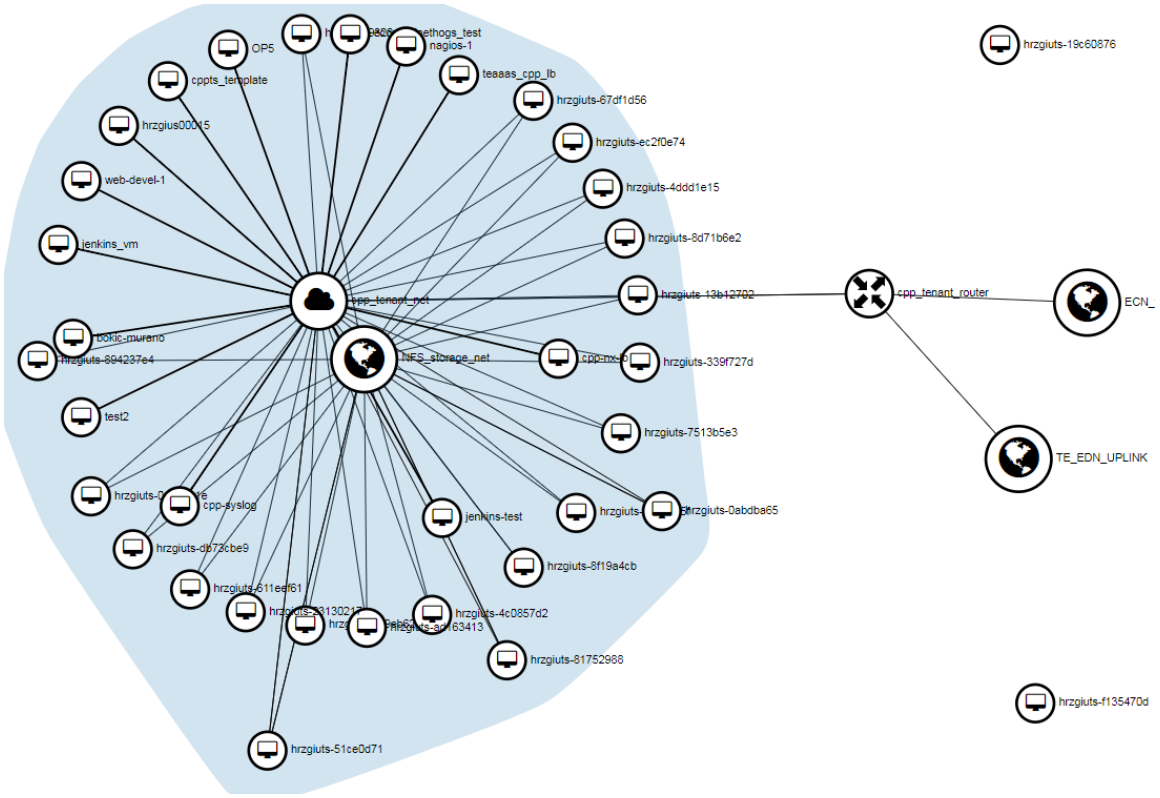
DHCP (*Dynamic Host Configuration Protocol*) poslužitelj dodjeljuje privatnu IP adresu mrežnom sučelju instance. U tom slučaju adresa je vidljiva pomoću naredbe poput `ip a`; dio je privatne mreže i koristi se za komunikaciju između instanci u istoj domeni emitiranja (*broadcast domain*) putem virtualne sklopke (L2 agent na svakom računalnom čvoru).

Preko usmjernika cpp_tenant_router mreža cpp_tenant_net ima izlaz na vanjsku mrežu ECN. Ta mreža predstavlja pogled na dio fizičke, vanjske mreže dostupne izvan OpenStack instalacije. IP adrese na vanjskoj mreži dostupne su od bilo koga na mreži izvan OpenStacka što koristimo prilikom spajanja na instance. Naime, na vanjskoj mreži alociramo plutajuće IP adrese (*floating IP address*) koje povezujemo s portovima na instancama.

Plutajuća IP adresa je usluga koju pruža Neutron. Kako na vanjskoj mreži DHCP nije

omogućen (pošto je vanjska mreža samo ”pogled” u svijet izvan OpenStacka), plutajuća IP adresa ne koristi DHCP usluge niti se statički postavlja unutar gosta. Instanca u biti nema saznanja da joj je dodijeljena plutajuća IP adresa. Isporuka paketa na sučelje s dodijeljenom plutajućom adresom je odgovornost Neutronovog agenta L3.

Load Balancer instanci je u oba tenanta dodijeljena plutajuća IP adresa. Zato prilikom mjerenja ”hrzgiuts” instancama pristupamo preko pripadnog Load Balancera. Nakon spajanja na Load Balancer preko njegove plutajuće adrese, putem interne mreže CPP_tenant_net, odnosno RSM_CI_DEV_ENV_net u drugom tenantu, spajamo se na ”hrzgiuts” instance koristeći njihovu privatnu IP adresu, što će biti vidljivo i u programskom kodu u sljedećem odjeljku.



Slika 4.22: Struktura mreže u OpenStack projektu CPP_tenant

Mjerenja u OpenStack projektima

Podaci koji se prikupljaju s instanci navedeni su tablici 4.15. To je u biti tablica metrics (4.8) iz baze big_data proširena komentarima o navedenim mjerenjima.

id	meter_id	unit	komentar
1	cpu_util	%	prosječno korištenje CPU-a
2	disk.write.bytes.rate	B/s	prosječna stopa pisanja na disk
3	disk.write.requests.rate	requests/s	prosječna stopa zahtjeva za pisanje na disk
4	disk.usage	MB	Fizička veličina spremnika *imagea* na domaćinu
5	memory.resident	MB	Količina RAM memorije koju koristi instanca na fizičkom stroju
6	memory.usage	MB	Količina RAM memorije koju koristi instanca od svoje dodijeljene memorije

Tablica 4.15: Vrste mjerenja na instancama u virtualnoj okolini

Kako OpenStack ima vlastiti telemetrijski servis, Ceilometer, koristimo ga kako bismo dobili mjerenja iz tablice 4.15. Ako Ceilometer na računalnom domaćinu (*compute hostu*) mjerene instance nije ispravno konfiguriran, Ceilometer će davati samo podatke o alociranim resursima, no ne i o njihovoj upotrebi. Zbog toga je potrebno na računalnom domaćinu u `/etc/ceilometer/pipeline.yaml` dodati `cpu_util` mjerenje, što za sobom povlači i ostala željena mjerenja iz tablice. No, to ne rješava problem dohvata mjerenja za `memory.usage` u CPP_tenatu. Na slici 4.23 vidimo da se `memory.usage` ne nalazi na popisu dostupnih mjerenja za instancu iz CPP_tenanta.

Razlog tomu je sljedeći - telemetrija se radi na hipervizorima, a instanca mora znati proslijediti podatke o resursima hipervizoru. Dakle, problem može biti u hipervizoru, instanci ili oboje. U službenoj dokumentaciji Ceilometra [8] navode se potrebni uvjeti za dohvat `memory.usage` mjerenja.

1. **libvirt verzija 1.1.1+** Libvirt je skup softvera koji uključuje biblioteku API-ja, servis libvirtd i alat komandne linije virsh. Primarni cilj libvirta je osigurati upravljanje s više različitih pružatelja virtualizacije (hipervizora) što uključuje i neke druge funkcionalnosti virtualizacije, kao što su upravljanje pohranom i mrežnim sučeljem. Instalirana verzija na računalnih domaćinima na kojima se nalaze instance CPP_tenanta je 1.2.9.3.

```
root@hrzgiclsctr088:~# ceilometer meter-list --query resource=4566e8fa-b3bb-4442-9346-ae6fa03d40ff
+--------------------------------+------------+------------+--------------------------------------+
| Name                           | Type       | Unit       | Resource ID                          |
+--------------------------------+------------+------------+--------------------------------------+
| compute.instance.booting.time  | gauge      | sec        | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| cpu                            | cumulative | ns         | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| cpu_util                       | gauge      | %          | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.allocation                | gauge      | B          | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.capacity                  | gauge      | B          | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.ephemeral.size            | gauge      | GB         | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.read.bytes                | cumulative | B          | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.read.bytes.rate           | gauge      | B/s        | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.read.requests             | cumulative | request    | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.read.requests.rate        | gauge      | requests/s | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.root.size                 | gauge      | GB         | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.usage                     | gauge      | B          | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.write.bytes               | cumulative | B          | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.write.bytes.rate          | gauge      | B/s        | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.write.requests            | cumulative | request    | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| disk.write.requests.rate       | gauge      | requests/s | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| instance                       | gauge      | instance   | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| memory                         | gauge      | MB         | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| memory.resident                | gauge      | MB         | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
| vcpus                          | gauge      | vcpu       | 4566e8fa-b3bb-4442-9346-ae6fa03d40ff |
+--------------------------------+------------+------------+--------------------------------------+
```

Slika 4.23: Popis dostupnih mjerenja za instancu u CPP_tenantu

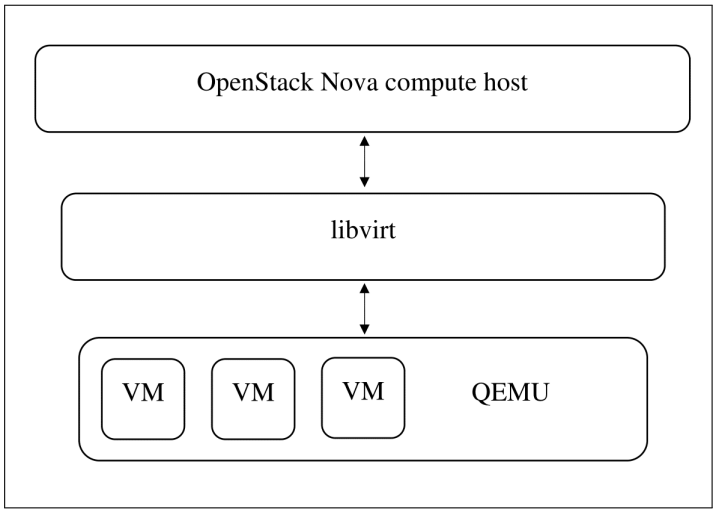
2. **QEMU verzija 1.5+** QEMU (kratko od Quick Emulator) je hipervizor otvorenog koda. Pripada hipervizorima tipa 2, onima koji se izvode na konvencionalnom operacijskom sustavu. U našem slučaju na svim računalnim domaćinima je instaliran operacijsk sustav Ubuntu 14.04.5. Instalirana verzija QEMU-a je 2.3.0. Upit za dobivanje libvirt i QEMU verzije na jednom od računalnih domaćina i odgovor prikazani su na slici 4.24.

3. **"Balon" upravljački program (*balloon driver*) za *image*** *Image* ili snimka od kojih je nastala instanca mora imati odgovarajući upravljački program za upravljanje radnom memorijom, što nije slučaj kod instanci u CPP_tenantu. Općenito, instance koje imaju "balon" upravljački program mogu označiti dijelove svoje RAM memorije kao neiskorištene (balonska inflacija). Hipervizor može osloboditi tu memoriju i koristiti ju za druge procese domaćina (*hosta*) ili za druge instance na tom domaćinu. Kada instanca kasnije zatreba oslobođenu memoriju, hipervizor ju opet dodjeljuje (deflacija balona) [9]. Prikaz strukture domaćin-hipervizor-gost vidljiv je na slici 4.25.

Na instancama iz RSM_CI_DEV_ENV tenanta nemamo taj problem jer su one nastale iz drugog *imagea*, tj. snimke na operacijskom sustavu CentOS pa podatke za `memory.usage` dohvaćamo na "konvencionalan" način od Ceilometra (funkcija `showCpu_util` u programskom kodu u nastavku). Kako instance iz CPP_tenanta zbog nedostajućeg "balon" upravljačkog programa ne znaju predati podatke o `memory.usage` hipervizoru na računalnom

```
[root@fuel2 ~]# ssh 192.168.99.9
Warning: Permanently added '192.168.99.9' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-101-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Wed Aug 30 10:28:40 2017 from 192.168.99.4
root@hrzgiclscpt070:~#
root@hrzgiclscpt070:~#
root@hrzgiclscpt070:~#
root@hrzgiclscpt070:~# virsh version
Compiled against library: libvirt 1.2.9
Using library: libvirt 1.2.9
Using API: QEMU 1.2.9
Running hypervisor: QEMU 2.3.0
```

Slika 4.24: libvirt i QEMU verzija na jednom od OpenStack računalnih domaćina



Slika 4.25: Prikaz strukture domaćin-hipervizor-gost

domaćinu na kojem se nalaze, te podatke dobivamo spajanjem SSH vezom na instancu preko Load Balancera (jedini način da s vanjskog servera izvan tenanta dođemo do instance) i pokretanjem naredbe `free -m`. U nastavku slijedi programski kod kojim se prikupljaju podaci sa ”produkcijskih” instanci CPP_tenanta. Podaci za RSM_CI_DEV_ENV tenant se prikupljaju na sličan način, na drugoj mreži i uz spomenutu razliku vezanu za `memory.usage`.

```
<?php
#egorpau
#2017.02.04
#derived from resources_get_from_ceilometer_get_nova_list1.php
header("Access-Control-Allow-Origin: *");
date_default_timezone_set('Europe/Zagreb');

class DB
{
    private static $db = null;

    private function __construct()
    {
    }
    private function __clone()
    {
    }

    public static function getConnection()
    {
        if (DB::$db === null) {
            try {
                DB::$db = new PDO("mysql: host=localhost; dbname=big_data_1;
                    charset=utf8", $user, $pass);
                DB::$db-> setAttribute(PDO::ATTR_ERRMODE,
                    PDO::ERRMODE_EXCEPTION);
            } catch (PDOException $e) {
                exit('PDO Error: ' . $e->getMessage());
            }
        }

        return DB::$db;
    }
}

function initializeMetersArray()
{
    $meters = array();

    $db = DB::getConnection();
    $st = $db->prepare('SELECT `id`, `meter_id` FROM metrics');
```

```
    $st->execute(array());

    while ($row = $st->fetch()) {
        $meters[$row['id']] = $row['meter_id'];
    }

    return $meters;
}

// lista aktivnih instanci od nxserver list
function getUsers()
{
    $output = `ssh ecoster\@172.19.125.92 sudo /usr/bin/nxserver --list |
        awk '{print $1, $3}'`;
    $list = explode("\n", $output);
    $addrsAndUsers = array();

    print_r($list);

    foreach ($list as $row) {
        // razdvoji redak rezultata na rijeci
        $values = preg_split('/\s+/', $row);
        // pogledaj prvu rijec, tj. prvi znak je li broj (prvi broj adrese)
        $first_char = $values[0][0];
        // ako je, spremi
        if (is_numeric($first_char)) {
            $addrsAndUsers[(string)$values[0]] = $values[1];
        }
    }
    print_r($addrsAndUsers);
    return $addrsAndUsers;
}

function getToken()
{
    $postData = array(
  'auth' => array(
    'tenantName' => 'CPP_tenant',
    'passwordCredentials' => array(
      'username' => 'username',
      'password' => 'password'
    )
```

```
  )
);

    // cURL setup
    $ch = curl_init('http://172.19.125.5:5000/v2.0/tokens');
    curl_setopt_array($ch, array(
  CURLOPT_POST => true,
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_HTTPHEADER => array(
    'Content-Type: application/json'
  ),
  CURLOPT_POSTFIELDS => json_encode($postData)
));
    // posalji zahtjev
    $response = curl_exec($ch);
    // provjeri greske
    if ($response === false) {
        die(curl_error($ch));
    }
    // dekodiraj odgovor
    $responseData = json_decode($response, true);
    // ispis podataka iz odgovora
    echo "\n TOKEN REST". $responseData['access']['token']['id'] . "\n";
    return $responseData['access']['token']['id'];
}


function showCpu_util($tokenR, $vmId)
{
    #mora se izvrsiti:#
    #ceilometer sample-list -l 1 --meter cpu_util -q
        'resource=f48f9945-57f3-466d-b21d-16593b2c7761'
    #curl -g -i -X 'GET'
        'http://192.168.91.2:8777/v2/meters/cpu_util?q.field=resource&q.op=eq&q.type=&q.
        -H 'User-Agent: ceilometerclient.openstack.common.apiclient' -H
        'X-Auth-Token: {SHA1}b29d5b3b60f3269cea4b8158d7d58b3782dca8dd'

    $rezultat = array();

    $meters = array("cpu_util", "disk.write.bytes.rate",
        "disk.write.requests.rate",
                "disk.usage", "memory.resident" );
```

```
    foreach ($meters as $meter) {
        $upit = "http://172.19.125.5:8777/v2/meters/" . $meter .
            "?q.field=resource&q.op=eq&q.type=&q.value=" .$vmId .
            "&limit=1";

        $ch = curl_init($upit);
        curl_setopt_array($ch, array(
    CURLOPT_POST => false,
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => array(
    'User-Agent: ceilometerclient.openstack.common.apiclient',
    "X-Auth-Token: $tokenR "
  )
));

        $response = curl_exec($ch);

        if ($response === false) {
            die(curl_error($ch));
        }
        // dekodiraj odgovor
        $responseData = json_decode($response, true);

        $vrijednost =
            number_format((float)$responseData[0]['counter_volume'], 2,
            '.', '');
        $rezultat[$meter] = $vrijednost;
    }
    return $rezultat;
}


function getNovaList($tokenR)
{
    #pokupi ID od cpp_ts-a sa adresom $ipAddr
    #potrebno pokrenuti:
    /*
    curl -g -i -X GET http://172.19.125.5:8774/v2.1/servers/detail -H
        "User-Agent: python-novaclient" -H "Accept: application/json" -H
        "X-OpenStack-Nova-API-Version: 2.25" -H "X-Auth-Token:
        gAAAAABYlclZvTDSiYnuCIeUjORlxC79VE8sPXmuJVHGpL3XZMVws--AbVQPTBdZy0_Orzs-UgAaIvVE
```

```
    */
    $ch = curl_init('http://172.19.125.5:8774/v2.1/servers/detail');
    curl_setopt_array($ch, array(
  CURLOPT_POST => false,
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_HTTPHEADER => array(
    'User-Agent: python-novaclient',
    'Accept: application/json',
    'X-OpenStack-Nova-API-Version: 2.25',
    "X-Auth-Token: $tokenR "
  )
));

    $response = curl_exec($ch);
    // Check for errors
    if ($response === false) {
        die(curl_error($ch));
    }
    // Decode the response
    $responseData = json_decode($response, true);
    $usersList = getUsers();
    $tenant = "6f7d2aa938ed40658ac22564aeee63d2";

    $db = DB::getConnection();
    foreach ($responseData['servers'] as $serverData) {
        $addr =
            (string)$serverData['addresses']['cpp_tenant_net'][1]['addr'];
        $host = (string)$serverData['OS-EXT-SRV-ATTR:hypervisor_hostname'];

        $user = "unknown";

        foreach ($usersList as $server => $user_name) {
            if ($addr == $server) {
                $user = $user_name;
                $table_instance_id;

                echo "\n ------------------------------------";
                echo "\nAdresa stroja:" . $addr ." \n";
                echo "\nUser je: " . $user . "\n";
                echo "\nHost je: " . $host . "\n";
                echo "\n instance_id " . $serverData['id'] . "\n";
```

```
$response = showCpu_util($tokenR, $serverData['id']);

// memory.usage dohvati direktnim spajanjem na stroj

$used_RAM = `ssh root@172.19.125.92 "ssh $addr free -m | awk
    'FNR == 2 {print \\\$3}'"`;
$memory_usage = intval(trim($used_RAM));
// print_r($memory_usage);
$response['memory.usage'] = $memory_usage;

try {
    $st = $db->prepare('SELECT id FROM instances WHERE
        instance_id =:instance_id');
    $st->execute(array('instance_id' => $serverData['id'] ));
    $row = $st->fetch();
    if (!$row) {
        try {
            $st = $db->prepare('INSERT INTO instances
                (instance_id, instance_id, tenant_id) VALUES
                (?, ?, ?)');
            $st->execute(array($serverData['id'], $addr, '1'));

            $instance_id = $db->lastInsertId();
        } catch (PDOException $e) {
            echo 'Greska: ' . $e->getMessage();
        }
    } else {
        $instance_id = $row["id"];
    }

    echo "\nID INSTANCE U TABLICI :". $instance_id ."\n ";
    // var_dump($row) ;
} catch (PDOException $e) {
    echo 'Greska: ' . $e->getMessage();
    break;
}

if ($user == "unknown") {
    $user_id = 0;
} else {
    try {
        $st = $db->prepare('SELECT id FROM users WHERE
```

```
            user_id = :ime_korisnika');
        $st->execute(array('ime_korisnika' => $user));
        $row = $st->fetch();
        if (!$row) {
            try {
                $st = $db->prepare('SELECT id FROM
                    users_openstack WHERE user_id =
                    :ime_korisnika');
                $st->execute(array('ime_korisnika' => $user));
                $row = $st->fetch();

                if (!$row) {
                    $st = $db->prepare('INSERT INTO
                        users_openstack (user_id) VALUES (?)');
                    $st->execute(array($user));

                    $user_id = $db->lastInsertId();
                } else {
                    $user_id = $row['id'];
                }
            } catch (PDOException $e) {
                echo 'Greska: ' . $e->getMessage();
            }
        } else {
            $user_id = $row['id'];
            echo "Id korisnika u bazi je ". $row['id'] . "\n";
        }
    } catch (PDOException $e) {
        echo 'Greska: ' . $e->getMessage();
    }
}


/*cpu_util", "disk.write.bytes.rate",
    "disk.write.requests.rate", "disk.usage",
    "memory.resident" */
echo '{
     "cpu_util": "' . $response['cpu_util'] . '",
     "disk.write.bytes.rate": "' .
        $response['disk.write.bytes.rate'] . '",
     "disk.write.requests.rate": "'.
        $response['disk.write.requests.rate'] .'",
```

```
                    "disk.usage": "' . $response['disk.usage'] . '",
                    "memory.usage": "' . $response['memory.usage'] . '",
                    "memory.resident": "' . $response['memory.resident'] .
                        '"
                }';
            echo "\n ---------------------------------- \n\n";

            $meters = initializeMetersArray();

            foreach ($response as $key => $value) {
                // Searches the array for a given value and returns the
                    first corresponding key if successful

                $meter_id = array_search($key, $meters);
                echo "meter_id $meter_id\n";

                if ($key == 'disk.usage') {
                    // bytes -> mb for disk.usage
                    $value /= 1048576;
                }
                $st2 = $db->prepare('INSERT INTO openstack_measuring
                    (instance_id, tenant_id, meter_id, value, user)
                    VALUES (?, ?, ?, ?, ?)');
                $st2->execute(array($instance_id, 1, $meter_id, $value,
                    $user_id));
            }
        } else {
            continue;
        }
    }
}

//pokupi token za REST
$tokenREST = getToken();
//echo "\n" . $tokenREST . "\n";
getNovaList($tokenREST);

exit();
```

Praćenje virtualne okoline pomoću Zabbixa

Zabbix [11] je softver otorenog koda dizajniran za IT poduzeća sa svrhom nadziranja servera, virtualnih strojeva i mrežnih uređaja koje poduzeće koristi kroz mjerenja potrošnje računalnih resursa u stvarnom vremenu. Podaci se spremaju u bazu podataka, a predstavljeni su putem grafičkog web sučelja o čemu brine serverska komponenta Zabbixa.

Postoji nekoliko opcija praćenja zadane mrežne okoline:

- Jednostavne provjere dostupnosti i odaziva standardnih usluga kao što su SMTP ili HTTP bez instalacije softvera na nadziranom računalu.
- Instalacija Zabbix Agenta na nadziranom računalu za praćenje statističkih podataka kao što su opterećenje procesora, korištenje mreže, prostora na disku i drugo.
- Zabbix uključuje i podršku za praćenje putem
  - SNMP (*Simple Network Management Protocol*)
  - TCP (*Transmission Control Protocol*)
  - ICMP (*Internet Control Message Protocol*)

  provjera, a uz prilagođene parametre i putem
  - IPMI (*Intelligent Platform Management Interface*)
  - JMX (*Java Management Extensions*)
  - SSH (*Secure Shell*)
  - Telnet

  Podržani su i različiti mehanizmi za slanje obavijesti u stvarnom vremenu, uključujući XMPP (*Extensible Messaging and Presence Protocol*).

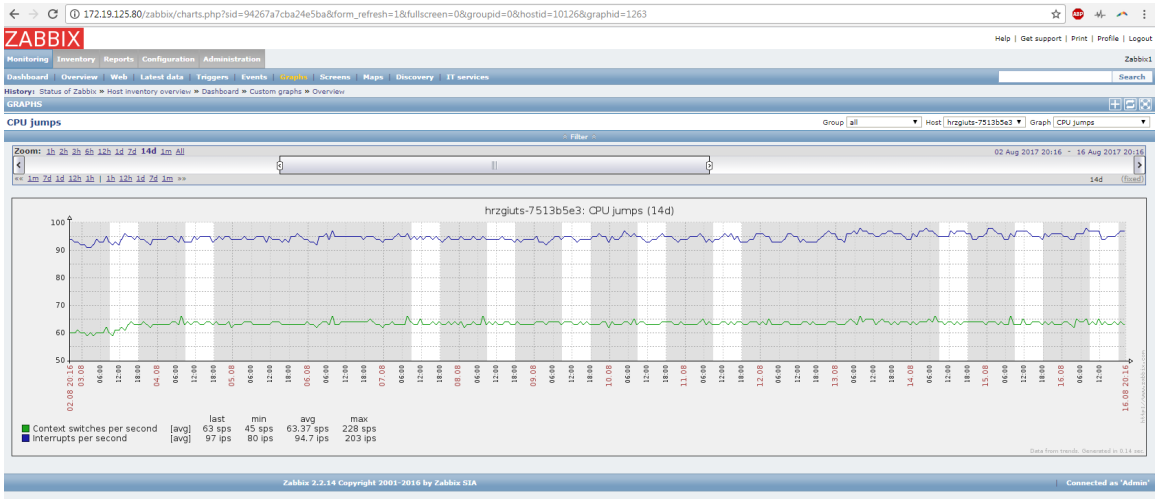
Radi usporedbe podataka dobivenih nad virtualnim okolinama RSM_CI_DEV_ENV i CPP_tenant na način opisan u prethodnoj sekciji 4.3 i korištenjem za to namijenjenog softvera, na novoj instanci u CPP_tenantu postavljamo Zabbix serversku komponentu, Zabbix Server 2.2.14. Ta instanca je nastala od iste snimke sustava kao i ”hrzgiuts” instance, ali ne preko Load Balancera već je ručno kreirana preko OpenStack nadzorne ploče Horizon. Riječ je, dakle, o instanci s operacijskim sustavom SUSE Linux Enterprise Server 11 SP1. Kako se na Zabbix serverskoj serverskoj komponenti nalazi programski kod za grafičko web sučelje, instanci je dodijeljena plutajuća IP adresa da bi se u web pregledniku lokalnog računala moglo pristupiti tom sučelju.

Za praćenje zadane mrežne okoline odabrali smo drugi pristup pa je tako na ”hrzgiuts” instance koje želimo pratiti instaliran je Zabbix agent. U agentovu konfiguracijsku datoteku upisuje se privatna adresa Zabbix servera kako bi mu instance mogle slati tražene podatke, a kako bi Zabbix server mogao automatski detektirati agenta i instancu koju treba pratiti (bez ručnog dodavanja na web korisničkom sučelju), u istoj konfiguracijskoj datoteci postavljamo `HostMetadataItem=system.uname`. Detaljan opis instalacije i naprednog konfiguriranja može se pronaći u [7].

Za dodane instance ”osnovni” agent nudi mjerenja za

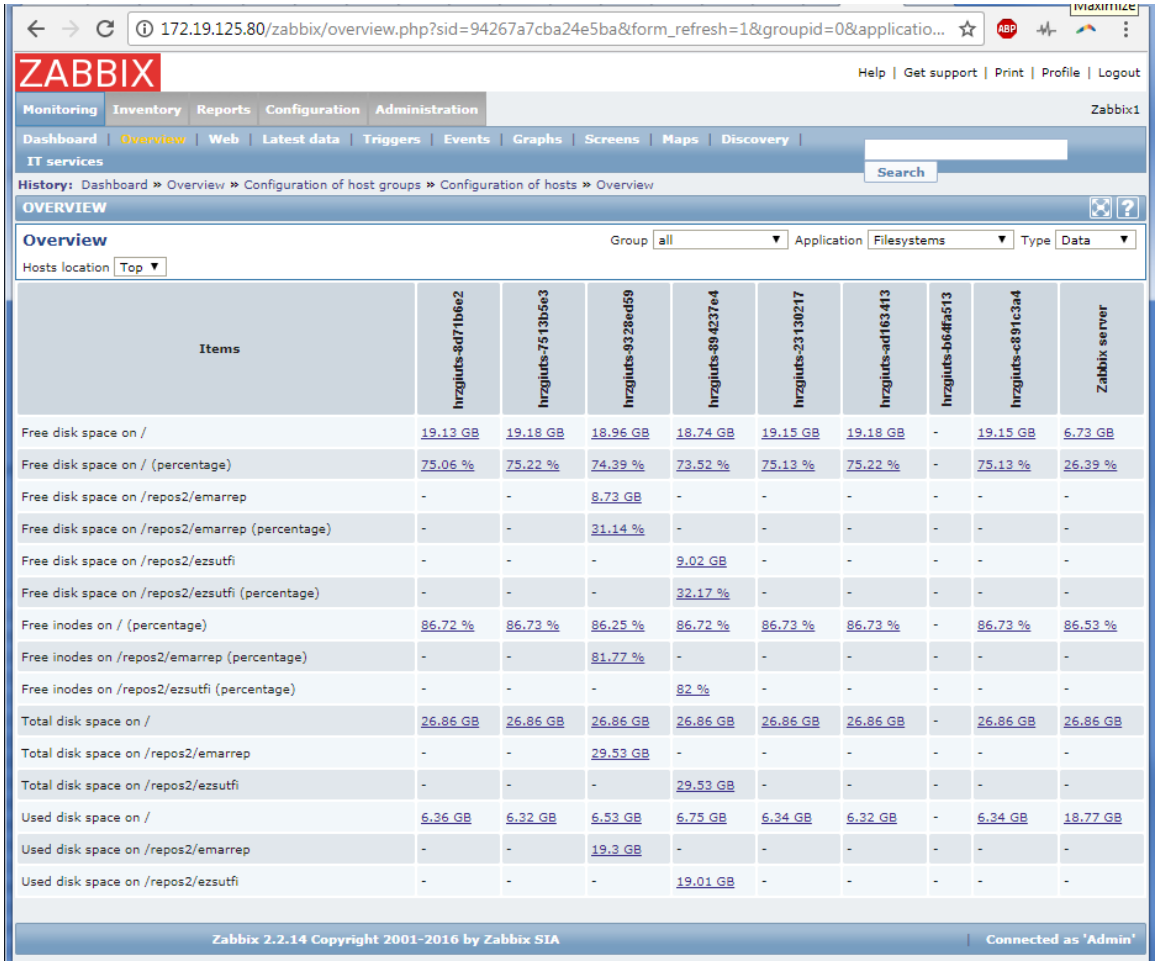
- CPU skokovi, opterećenje, iskorištenost
- upotreba radne memorije, *swap* memorije, prostora na disku
- mrežni promet na eth0, eth1

Prikaz nekih od navedenih mjerenja (upotreba diska i CPU skokovi) može se vidjeti na slikama 4.26 i 4.27.



Slika 4.26: Graf CPU skokova za odabranu instancu unutar zadnjih 14 dana

Struktura Zabbix baze podataka je kompleksna i upiti za vrijednosti pojedinih mjerenja zahtijevaju njeno dobro poznavanje kako bi se dobile prave vrijednosti u željenom vremenskom razdoblju što čini ovaj pristup praćenja instanci nezgodnim ako želimo samo obrađivati prikupljene podatke. U verziji 3.4. je pristup ”sirovim” podacima ipak olakšan,



Slika 4.27: Prikaz upotrebe prostora na disku po instancama i datotečnim sustavima

što u budućnosti može promijeniti način pristupa problemu daljnje obrade dobivenih podataka, no i s trenutno instaliranom verzijom 2.2 uvid u stanje cjelokupne praćene okoline korektan i funkcionalan te može biti dodatna pomoć administratorima CPP_tenant i RSM_CI_DEV_ENV virtualnih okolina.

Zaključak

Mjerenje projektnih diskova iz tablice 4.12 administratorima servera na kojima se ti diskovi nalaze omogućuje brz i jednostavan uvid u promjene u veličini upotrebljenog prostora. Korisnicima Cloud Web Portala omogućuje uvid koliko trenutno prostora zauzimaju od onog koji im je dodijeljen na pripadnom home direktoriju.

Spremanje izmjerenih podataka u MySQL bazu stvara dobar temelj za daljnju obradu podataka u svrhu predviđanja upotrebe diskovnog prostora u nekom budućem vremenskom razdoblju (duže razdoblje prikupljanja podataka - veći vremenski interval predviđanja). Trend linearnog rasta ukazao je na popunjenje kapaciteta tri od deset mjerenih diskova u idućih pola godine.

Također, vizualizacijom podataka uočeno je da dosta korisnika ima malu razliku između najmanje i najveće zabilježene vrijednosti mjerenja zazuzeća diskovnog prostora u pripadnom home direktoriju, što ukazuje na slabu aktivnost pisanja i brisanja.

U međuvremenu, tijekom pisanja ovog rada, ukazala se potreba za još jednom tablicom stoga je u big_data bazu podataka dodana nova tablica particije_prema_userima u kojoj se za svakog korisnika koji koristi prostor na diskovima iz tablice particije bilježi koliko on iznosi. Ideja je te podatke prikazivati u obliku kružnog grafa unutar "User details" sekcije na Cloud Web Portalu, kako bi korisnik vidio na kojim sve lokacijama zauzima koju količinu prostora.

Cilj mjerenja na OpenStack virtualnim okolinama bio je, uz isporuku podataka trećoj strani, odrediti najprikladniji način dohvaćanja podataka o potrošnji računalnih resursa. Navodimo isprobane opcije i njihove prednosti i nedostatke.

- **Dohvat podataka od OpenStackovog servisa Ceilometer**. Ceilometer daje korektne podatke, no moguća su nepredviđena ponašanja. Naime, automatsko ažuriranje ili nešto slično "pogazi" postavke u Ceilometerovim konfiguracijskim datotekama i onemoguće se mjerenja, što nam se i dogodilo jednom prilikom, a kao posljedicu ima "rupu" u podacima u bazi. Također, mjerenja ovise i o operacijskom sustavu virtualnog stroja kojeg želimo pratiti pomoću Ceilometra, a uzroci i posljedice ovog nedostatka opisani su u 4.3. U budućnosti će virtualni strojevi u CPP_tenantu vjero-

jatno imati noviju verziju operacijskog sustava SUSE Linux Enterprise Server što će riješiti problem dohvata mjerenja za `memory.usage`.

- **Izravno spajanje na virtualni stroj i pokretanje odgovarajućih naredbi komadne linije**. Ovim pristupom dobivamo sve željene podatke, uz uvjet da su postavljeni SSH ključevi. Za spremanje mjerenih podatak u bazu ovim pristupom možemo dobiti što trebamo, no za trenutne upite i prikaz podataka na Cloud Web Portalu to predugo traje.

- **Korištenje sustava za nadziranje treće strane**. U našem slučaju riječ je o sustavu Zabbix koji daje dobar pregled nad cjelokupnim stanjem sustava koji se mjeri, no trenutna instalirana verzija nije pogodna za dohvat "sirovih" podataka. Za Zabbix server je planirano prebacivanje u novi OpenStack tenant koji će sadržavati administratorske alate. Predložena je nova instalacija na zadnju verziju pošto je trenutna instalacija testna s ciljem istraživanja mogućnosti Zabbixa u danoj okolini. Također je planirana nove snimka sustava od koje će Load Balancer i Nova servis stvarati nove instance u pripadnom tenantu. Snimka bi sadržavala konfigurirani Zabbix agent usmjeren prema novom Zabbix Serveru pa bi se tako uz CPP_tenant pratila i virtualna okolina RSM_CI_DEV_ENV.

S obzirom na trenutno instalirane verzije OpenStacka, operacijskih sustava na virtualnim strojevima u OpenStack okolinama i verziju Zabbixa, svaki od ovih pristupa je najbolje iskorišten za pojedinu svrhu - prikupljanje podataka u svrhu daljnje analize, prikaz na Cloud Web Portalu i administraciju sustava, a gore opisana poboljšanja će pokazati jesu li potrebne promjene u pojedinom pristupu s ciljem veće točnosti, brzine i efikasnosti.

Bibliografija

[1] P. DuBois, *MySQL, Fifth Edition*, Addison-Wesley Professional, 2013.

[2] T. Erl, R. Puttini i Z. Mahmood, *Cloud computing: concepts, technology & architecture*, Pearson Education, 2013.

[3] E. S. Gardner, *Exponential smoothing: The state of the art*, Journal of forecasting **4** (1985), br. 1, 1–28.

[4] Y. Jadeja i K. Modi, *Cloud computing-concepts, architecture and challenges*, Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on, IEEE, 2012, str. 877–880.

[5] R. Manger, *Baze podataka*, Element, 2012.

[6] P. Mell, T. Grance et al., *The NIST definition of cloud computing*, (2011).

[7] R. Olups, *Zabbix 1.8 network monitoring*, Packt Publishing Ltd, 2010.

[8] OpenStack, *OpenStack Ceilometer Measurements*, `https://docs.openstack.org/ceilometer/latest/admin/telemetry-measurements.html`, travanj 2017.

[9] RedHat, *Balloon driver*, `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/Technical_Reference_Guide/sect-Technical_Reference_Guide-Virtualized_Hardware-Balloon_driver.html`, kolovoz 2018.

[10] O. Sefraoui, M. Aissaoui i M. Eleuldj, *OpenStack: toward an open-source solution for cloud computing*, International Journal of Computer Applications **55** (2012), br. 3, 38–42.

[11] A. Vladishev, *Zabbix Documentation*, `https://www.zabbix.com/documentation/2.2/`, svibanj 2017.

Sažetak

Pojam računanja u oblaku prisutan je u gotovo svim IT poduzećima, uključujući i tvrtku Ericsson Nikola Tesla koja bazira interni projekt TEAaaS (Test Environment and Applicaton as a Service) na OpenStack operacijskom sustavu u oblaku. TEAaaS omogućuje korisnicima unutar tvrtke virtualne strojeve za razvoj i testiranje aplikacija. Na tim virtualnim strojevima raspoređenima u dvije virtualne okoline radimo mjerenja potrošnje računalnih resursa koristeći OpenStack servis Ceilometer, izravno spajanje na virtualni stroj i sustav za nadgledanje Zabbix. Na projektnim diskovima na fizičkim serverima podatke o zauzeću koristimo za prikaz na TEAaaS web portalu i predviđanje budućih vrijednosti što pokazuje punjenje kapaciteta tri od deset mjerenih diskova unutar idućih pola godine. Također, dobivamo uvid u korisnička ponašanja na njima dodjeljenim `home` direktorijima.

Summary

The cloud computing concept is present in almost all IT companies, including Ericsson Nikola Tesla. Its internal project TEAaaS (Test Environment and Appliance as a Service) is based on the OpenStack cloud operating system. TEAaaS provides users within the company with virtual machines for developing and testing applications. On these virtual machines deployed in two virtual environments, we measure the consumption of computing resources using the OpenStack Ceilometer service, direct connection to the virtual machine and the Zabbix monitoring system. On project disks on physical servers, we use the used disk space data for visualization on the TEAaaS web site and for prediction of future data values. These measurings indicate a capacity charge of three of the ten metered disks. Also, we are getting insight into user behavior on `home` directory assigned to user.

Životopis

Rođena sam 13. ožujka 1993. u Vinkovcima, gdje sam pohađala Gimnaziju Matije Antuna Reljkovića (opći smjer) i Srednju glazbenu školu Josipa Runjanina (smjer glazbenik klavirist). 2011. godine sam upisala Preddiplomski sveučilišni studij Matematika na matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu, a 2015. diplomski studij Računarstvo i matematika. Tijekom ljeta 2016. godine sudjelovala sam na Ericsson Summer Campu na projektu TEAaaS koji je opisan u ovom radu. Preciznije, radila sam s OpenStack Murano servisom gdje sam aplikacijske pakete predviđene za brzu i jednostavnu implementaciju u OpenStack okolini prilagođavala Ericssonovoj mrežnoj okolini. Nakon Summer Campa na istom odjelu (ITTE) sam nastavila rad s OpenStack okolinom.