Sistem za upravljanje klaudom korišćenjem REST principa

Dušan Kilibarda, dr Dušan Vujošević

***Sadržaj* — U ovom radu je predstavljena arhitektura sistema za upravljanje klaud infrastrukturom koji koristi REST principe i REST API za komunikaciju između distribuiranih sistemskih modula. Objašnjavaju se prednosti i mane predstavljenih sistema i načina komunikacije sa klaud platformom na primeru Klaudsteka. Navode se primeri njihove primene u realnom okruženju. Ideja za ovakvu arhitekturu je rezultat lične želje da se prevaziđu konceptualne greške koje su stvarale praktične probleme i ograničenja u pružanju usluga tokom i nakon dvogodišnjeg razvoja komercijalne platforme za upravljanje klaud resursima na kojoj je rađeno.**

***Ključne reči* —Klaud arhitektura, REST, softverska arhitektura, distribuirani sistemi, sistemi bez stanja, klaud menadžment platforma**

I. UVOD

SVE brži razvoj tehnologije doprinosi sve bržem razvoju aplikativnog softvera od ideje do gotovog proizvoda. Kako se broj aplikacija povećava, tako se i načini monetizacije menjaju, pa je danas jako zastupljena ideja da se korisnicima pruži besplatan softver, uz koji će im se povremeno plasirati reklame. Kvalitetan besplatan softver privlači veliki broj korisnika, što direktno utiče na arhitekturu softvera i kao zahteve nameće visoku dostupnost servisa i kratko vreme čekanja korisnika na odgovor od strane servera kako bi aplikacija ostala konkurentna u moru drugih aplikacija. Ovakav pristup razvoju aplikacija je u poslednjih nekoliko godina postao standard, pa se mali timovi (startapovi), koji najčešće nemaju dovoljno finansijskih sredstava da investiraju u fizičku serversku infrastrukturu, odlučuju za zakup infrastrukture za svoju aplikaciju. Trenutno na tržištu postoje klaud provajderi koji svoju infrastrukturu nude drugima na korišćenje

Dušan Kilibarda, Računarski fakultet, Beograd, Srbija (e-mail: dusankil@gmail.com)
dr Dušan Vujošević, Računarski fakultet, Beograd, Srbija (e-mail: dvujosevic@raf.rs)

kroz svoje klaud platforme. Problem sa velikim brojem korisnika je i taj što se ti korisnici često ne nalaze na istoj geografskoj lokaciji, pa je vrlo verovatno da kod udaljenih korisnika postoji kašnjenje u usluzi. Klaud provajder ne može da reši ovaj problem osim ukoliko nema geografski distribuiranu infrastrukturu.

Ideja ovog rada je da predstavi sistem za upravljanje klaud resursima koji je potpuno nezavisan od klaud platformi koje koristi, tako da može koristiti usluge više distribuiranih klaud provajdera širom sveta, a da korisniku pruža istu uslugu i privid da se sve vreme radi o istoj platformi i istom klaud provajderu.

U nastavku rada se ističe značaj REST principa u postizanju ovog cilja, kao i značaj uspostavljanja distribuirane multiservisne arhitekture. Takođe se predstavljaju principi rada sistema i njegove komponente.

II. REST Principi i Arhitektura

Representational State Transfer (REST) je tip mrežne sofrverske arhitekture nastao još tokom razvoja Hypertext Transfer Protocol-a (HTTP), koji se postiže poštovanjem osnovnih REST principa, a koji za cilj ima pravilnu upotrebu HTTP prokokola unutar distribuiranih sistema. Poštovanjem REST principa se omogućava kreiranje sistema koji mogu da komuniciraju putem interneta i lako skaliraju po potrebi.

REST principi su:

- Svaki resurs je određen uniformnim identifikatorom - Uniform Resource Identifier (URI), koji se koristi za pristup samom resursu
- Resursima se pristupa korišćenjem HTTP zahteva: POST, GET, PUT, DELETE
- Svaki zahtev mora da sadrži sve informacije potrebne za njegovo izvršavanje

Prvi princip omogućava da se svi resursi predstave na uniforman način i da se svakom resurslu jednostavno pristupa.

Drugi princip promoviše samo četiri HTTP zahteva za pristup resursima. GET zahtev za islistavanje kolekcija ili čitanje pojedinačnog resursa. POST za kreiranje novog resursa. PUT za izmenu postojećeg resursa ili kolekcije. DELETE za brisanje resursa ili kolekcije. GET, PUT i DELETE zahtevi su idempotentni jer su rezultati koji su posledica njihovog izvršavanja uvek isti. GET ne menja resurse, DELETE će obrisati resurs ili kolekciju ukoliko resurs ili kolekcija postoje, nakon izvršenja resurs ili kolekcija neće postojati. PUT zahtev ima dvojako ponašanje i dvojaku primenu. Ukoliko se PUT

koristi za izmenu nepostojećeg resursa, resurs će biti kreiran sa prosleđenim parametrima. Ukoliko resurs već postoji, biće izmenjen tako da odgovara prosleđenim parametrima – u svakom slučaju, resurs će postojati u obliku u kome je prosleđen PUT zahtevom. Ukoliko korisnik želi da kreira resurs koji je već postojao, POST metod bi mu vratio grešku, dok bi PUT metod izmenio resurs i vratio pozitivan odgovor, što u nekim slučajevima više odgovara logici sistema, pa se i za kreiranje i za izmenu koristi PUT.

Treći princip predviđa da deo sistema koji izvršava traženu operaciju ne čuva stanje resursa između operacija (sistem je stateless [2]) i da svi parametri potrebni za uspešno izvršavanje operacije nad resursom budu dostavljeni u istom HTTP zahtevu. Na ovaj način se postiže da više različitih instanci istog servera u istom trenutku može da vrši istu funkciju, a da ne zavise jedna od druge, što povećava broj korisničkih zahteva koji može biti obrađen u isto vreme.

Poštovanjem navedenih principa prilikom razvoja aplikativnih modula postiže se laka skalabilnost jer je pristup modulima uniforman, moduli su nezavisni jedni od drugih, informacije dostavljene u zahtevu su dovoljne za uspešnu realizaciju zahteva i korisnik nije vezan za jedan određeni server već može koristiti drugi server za svaki sledeći zahtev, a da pritom odgovor koji dobije bude isti. Takođe moduli koji međusobno komuniciraju i poštuju REST principe, mogu biti nezavisno razvijani u različitim tehnologijama, mogu biti potpuno zamenjeni u bilo kom trenutku, a da to ne poremeti funkcionisanje sistema. Zbog svega navedenog i sve veće popularnosti servisa čiji sadržaj generišu korisnici, REST je dobio veliku popularnost i postao okosnica WEB2.0[1] servisa.

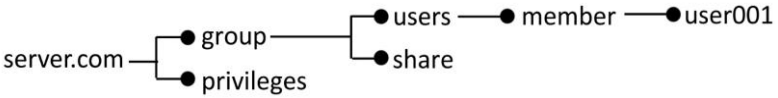
III. REST API

Aplikacije koje poštuju REST principe najčešće su podeljene na klijentski i serverski deo. Ova podela je veoma bitna jer je uvek klijent taj koji inicira komunikaciju, dok server uvek odgovara na već postojeći zahtev. Serverski deo može biti centralizovan ili distribuiran i izvršava se na jednom ili više servera, dok je klijentski deo aplikacije distribuiran na korisničke mašine. Distribuirana serverska aplikacija može biti podeljena na module po nameni ili na neki drugi način. Modulima se može pristupati direktno ili se može odrediti jedan ili više pristupnih modula koji služe da dalje komuniciraju sa drugim modulima unutar sistema, zaduženim za obradu dobijenog zahteva.

[1] WEB2.0 je termin nastao 1999. godine, a odnosi se na novu generaciju internet servisa koji više nisu koncipirani na podeli uloga na one koji kreiraju i one koji koriste sadržaj, već sami korisnici kreiraju sadržaj na servisima i dele ga na internetu [3]

A. REST API komunikacija

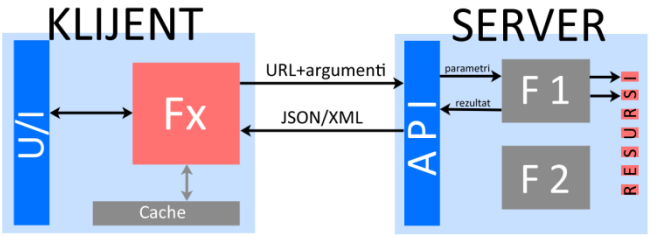
Klijentski i serverski delovi aplikacije komuniciraju putem javne ili privatne mreže, koristeći HTTP zahteve unutar HTTP ili HTTPS protokola. HTTP/HTTPS zahtev sadrži Uniform Resource Locator (URL) i parametre potrebne za izvršenje zahteva. URL predstavlja adresu, koja određuje resurs nad kojim bi zahtev trebalo da se izvrši.



Sl. 1. Koncept hijerarhije resursa

Sam koncept hijerarhije resursa direktno utiče na izgradnju URI, odnosno URL identifikatora. Za postojeći primer na Sl. 1, resursu "user001" bi se pristupalo koristeći URL http://server.com/group/users/member/user001. Zavisno od vrste zahteva koji bi se koristio, bilo bi moguće pročitati parametre resursa, naptaviti u njemu novi resurs, obrisati ga ili izmeniti neki njegov parametar.

Skup ovako organizovanih resursa kojima se pristupa na prethodno definisan način korišćenjem URL-a, naziva se stablo resursa. Čvorovi stabla resursa predstavljaju logičke ili funkcionalne grupe, a listovi predstavljaju podatke ili promenjive. Skup procedura i funkcija koje služe za pristup podacima, aplikacijama i sustemskim funkcijama na nekom sistemu, koji vraća unapred definisani tip podataka, naziva se Application Programming Interface (API). Ukoliko se API dodatno pridržava REST principa, u pitanju je REST API.

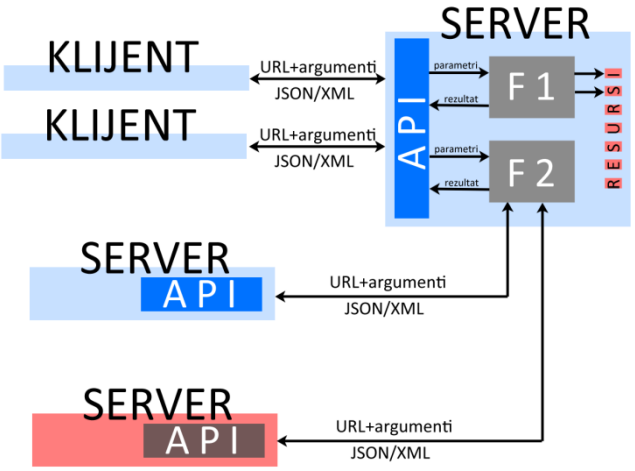


Sl. 2. Komunikacija između krajnjeg korisnika i sistema

Na Sl. 2 se vidi proces komunikacije između krajnjeg korisnika koji u ovom slučaju ima ulogu klijenta i sistema koji ima ulogu servera. Korisnik kroz U/I (korisnički interfejs) inicira Fx funkciju, koja kreira korisnički

zahtev i šalje ga API funkciji na serveru. API je u ovom slučaju skup sistemskih funkcija koje su iz perspektive servera date na korišćenje celom spoljnom svetu. Takođe API funkcije su mapirane na F1 i F2 funkcije unutar modula, koje služe za operacije nad resursima. Poruka o ishodu funkcije se istim putem vreća korisniku u vidu HTTP odgovora.

Na isti način mogu komunicirati i dva sistemska modula, iako su oba modula deo istog sistema, modul koji inicira komunikaciju će biti klijent, dok će modul koji odgovara na zahtev biti server.



Sl. 3. Komunikacija unutar serverskih modula

Na Sl. 3 je prikazan scenario gde dva različita sistemska modula u ulozi klijenta iniciraju komunikaciju sa trećim modulom koji dobija ulogu servera. Funkcija F1 vrši izmene nad lokalnim resursima, dok funkcija F2 može inicirati komunikaciju sa API interfejsima lokalnih ili udaljenih (crveno na slici) servera. U ovom slučaju server sa funkcijama F1 i F2 u isto vreme ima i ulogu klijenta i ulogu servera, što je vrlo često u distribuiranim sistemima koji koriste REST. Važno je primetiti u ovom primeru da se komunikacija sa eksternim sistemima odigrava na isti način, korišćenjem REST API-a.

B. Pozitivni i negativni uticaji REST API-a

Ovakav pristup omogućava laku integraciju novih modula u već postojeći sistem, oslanjajući se pritom na skalabilnost kroz već definisane REST principe. Osim skalabilnosti i lakše integracije dodatnih modula, ovakva arhitektura neretko obezbeđuje bolje performanse sistema i mnogo lakšu kasniju administraciju, održavanje koda i dokumentacije.

Negativan uticaj REST API-a na sistem se ogleda u povećanom korišćenju mrežnih resursa. Uzrok je veliki broj ponovljenih HTTP zahteva, najčešće GET zahteva za dobijanje informacija o resursu. Na ove zahteve sistem šalje uvek isti odgovor dok god se stanje resursa ne promeni. Delimično rešenje za ovaj problem je čuvanje dobijenih podataka u klijentskoj aplikaciji (keširanje podataka). Ipak nije u potpunosti moguće ukloniti slanje suvišnih zahteva, ali je uz dobru kontrolu moguće svesti ih na razumnu meru. REST sistem ima implementiran mehanizam za kontrolu kešinjanja - unutar odgovora koji se šalje klijentu šalje se poseban tag, koji označava da li je keširanje za određene podatke dozvoljeno ili nije.

Uniformni interfejsi koje koristi REST API mogu imati lošije performanse od interfejsa pisanih za tačno određenu funkciju. Ovaj nedostatak je često uzrokovan većom raslojenošću softverskog sistema koji koristi REST API. Tako na primer sloj za parsiranje API zahteva na serveru ne može da vrši direktne promene na resursu, već mora da se obrati sloju ispod sebe, koji zahtev za promenu prosleđuje sledećem sloju, dok se ne stigne do sloja koji je namenjen za izmenu resursa. Tako je svaki sloj unutar softvera svestan samo sloja ispod i iznad sebe, što utiče na povećanje vremena izvršavanja zahteva, ali istovremeno sistem čini bezbednijim i lakšim za dalji razvoj i održavanje.

S obzirom na to da je REST predviđen za implementaciju internet softverskih servisa, što predviđa veliku količinu podataka, korisnika i funkcija, za koje bi morali da budu napisani posebni interfejsi, korist koja se dobija uniformnim interfejsima je u najvećem broju slučajeva veća od nedostataka koje ovi interfejsi imaju.

Modularna arhitektura koja koristi REST takođe omogućava mnogo bolju kontrolu resursa, pa se određeni moduli aktiviraju samo ukoliko se zahteva neka od njihovih funkcionalnosti - u suprotnom ne troše skoro nikakve resurse. Ukoliko dođe do prekida rada nekog modula, moguće ih je vrlo lako i brzo zameniti, a ostali moduli pritom nastavljaju da rade nesmetano.

Kako REST koristi komunikaciju u vidu zahteva i odgovora, gde klijent uvek šalje zahtev, postoji problem prilikom izvršenja asinhronih funkcija, kada nije praktično čekati odgovor nekog servera, jer trajanje izvršenja zahteva nije poznato. U tom slučaju se može iskoristiti poseban modul unutar sistema koji bi imao jednu jedinu ulogu, a to je da inicira zahtev ka korisniku. Ovo bi bila jedina tačka u sistemu koja krši REST principe, ali je najpraktičnije rešenje i pozitivan doprinos koji daje u brzini obaveštavanja korisnika i smanjenju resursa potrebnih za druge načine rešavanja ovog problema opravdava kršenje arhitekture u ovom slučaju.

C. REST-like sistemi

Neki sistemi koriste samo određene REST principe, pa se neretko sreću sistemi koji je umesto svih nabrojanih HTTP zahteva koriste samo POST i GET zahteve. Zbog ovog ograničenja, funkcionalnosti PUT i DELETE zahteva se moraju implementirati korišćenjem postojećih POST i GET zahteva u kombinaciji sa dodatnim parametrima unutar samog zahteva. To znači da API parser mora imati i dodatnu funkcionalnost, da osim vrste zahteva prepozna i traženi metod, odnosno funkciju koju je klijent želeo da pokrene. To u suštini i nije loša promena, jer sada osim četiri osnovne funkcije možemo da definišemo nebrojeno mnogo različitih funkcija. Ipak interakcija drugih REST sistema sa ovim sistemom bi zahtevala dodatnu implementaciju i ne bi bila u potpunosti RESTful.

IV. KLAUD PLATFORMA – KLAUDSTEK

Klaudstek (CloudStack) je besplatna, otvorena klaud menadžment platforma koja sadrži skup alata za manipulisanje resursima unutar klaud sistema. Osnovna namena je pružanje Infrastructure-as-a-Service (IaaS) usluge. Konkurenti na tržištu su mu brojni, a neki od poznatijih su OpenStack, Amazonov Elastic Compute Cloud (EC2), Majkrsofrov Azure, HP-ov Eucalyptus…

Osim što ima komplet alata za implementaciju i upravljanje IaaS, Klaudstek ima i veoma dobar korisnički interfejs za administraciju, koji se oslanja na Klaudstek REST-like API. U ovom radu je Klaudstek odabran kao primer jer već poseduje REST-like API, ali hipotetički na njegovom mestu može biti bilo koja klaud platforma, odnosno bilo koj REST API koji upravlja lokalnim ili udaljenim klaud resursima.

A. Klaudstek API interfejs

Ova klaud platforma ima predefinisane nivoe pristupa API interfejsu, koji određuju kom skupu komandi se može pristupiti. Nivoi pristupa su u vidu podskupova, tako da „Root admin“ ima pristup svim komandama za upravljanje virtuelnim i fizičkim resursima, „Domain admin“ ima pristup komandama za upravljanje virtuelnim resursima na nivou administratorskog domena, dok „User“ kao najniži nivo pristupa ima pristup komandama za upravljanje korisničkim instancama, skladištima podataka i mrežom.

API zahtevi se prosleđuju korišćenjem HTTP POST i GET zahteva u kojima se prosleđuju komanda i argumenti potrebni za izvršavanje komande. Komande se dele na dve vrste, komande kojima se vrše promene na serveru i komande kojima se samo izlistavaju potrebne informacije sa servera. Takođe je moguće koristiti HTTPS umesto HTTP protokola. Osnovna pristupna tačka za Klaudstek API je http://server.com:8080/clinet/api.

B. Struktura API zahteva

Tipičan primer Klaudstek api poziva izgleda ovako: http://localhost:8080/client/api?command=nekaKomanda&argument1=1&argument2=2&argument3=3&apiKey=zKFXJPCSBLoesn14SOlUHxcbyzUoEQ1VN5_Ww4GCcsiFHbTaIZ1uwR2rEcAXoBeBObrWD2I_EGCGoCjgJP6StQ&signature=Lxx1DM40AjcXU%2FcaiK8RAP0O1hU%3D

Primećuje se da se osim argumenta „command“ i argimenata potrebnih zaizvršavanje komande „argument1, argument2“ u linku nalaze i argumenti „apiKey“ i „signature“. Ovo su takođe neizostavni elementi svakog Klaudstek API zahteva, a služe za autentifikaciju i autorizaciju kreatora API zahteva. API ključ (apiKey) se kreira na klaud platformi i prosleđuje se korisniku prilikom autentifikacije, a korisnik ga onda prosleđuje u svakom zahtevu umesto korisničkog imena i šifre. Na ovaj način se bar delimično otežava krađa korisničkih pristupnih parametara. Potpis API zahteva (signature) se kreira prilikom kreiranja API zhteva i jedinstven je za svaki zahtev. Svrha njegovog postojanja je potvrda integriteta zahteva, kako parametri koji se prosleđuju ne bi bili izmenjeni u toku prenosa preko interneta. Način kreiranja potpisa je objašnjen u dokumentaciji Klaudstek API-a [4].

Trajanje API ključa, broj zahteva koji će platforma primiti od korisnika i broj zahteva koji će vratiti takođe mogu biti argumenti API zahteva. Oblik u kome će Klaudstek vratiti odgovor isto može da se definiše unutar API zahteva kroz argument response. Podržani oblici su Extensible Markup Language (XML) i JavaScript Object Notation (JSON). Ukoliko argument nije postavljen, odgovor od Klaudsteka će stići u obliku XML-a. Razlika između ovih formata je u načinu parsiranja. XML elemente predstavlja u obliku tagova <ime_taga>vrednost</ime_taga>, dok JSON koristi key:value parove { “ime_taga” : “vrednost” }. Iako je XML lakši za razumevanje, JSON je brži za parsiranje zbog manje količine podataka koja se prenosi.

Klaudstek ima razvijen i sistem za upravljanje greškama, pa tako ukoliko API zahtev ne može da se izvrši, Klaudstek će vratiti podatak o grešci u vidu koda, kao i kratak tekstualni opis greške. Kodovi grešaka se mogu naći u dokumentaciji Klaudstek API-a [4].

V. SISTEM ZA UPRAVLJANJE KLAUDOM

Jedan od zahteva klauda je potpuna automatizacija akcija tako da se svaki korisnički klik završi željenim rezultatom bez dodatne akcije od strane administratora. Iako Klaudstek već ima svoj grafički korisnički interfejs, zbog nedostatka automatizacije, nedovoljne kontrole nad privilegijama korisnika i nemogućnosti integracije eksternih modula, a samim tim i manjka

funkcionalnosti, ne uklapa se u željeni sistem. Zbog toga je potrebno klaud platformu integrisati u sistem koji ima sve dodatne module potrebne za procese isporuke, održavanja i naplate klaud servisa. [5]

Cilj sistema za upravljanje klaudom u koji bi klaud platforma bila integrisana je da korisniku pružu univerzalno korisničko iskustvo, usluge i prateće alate koji bi bili potpuno isti, bez obzira da li se koristi neka određena klaud platforma, više klaud platformi, privatni klaud korisnika ili javni klaud nekog klaud provajdera.

VI. Komponente Sistema Za Upravljanje Klaudom

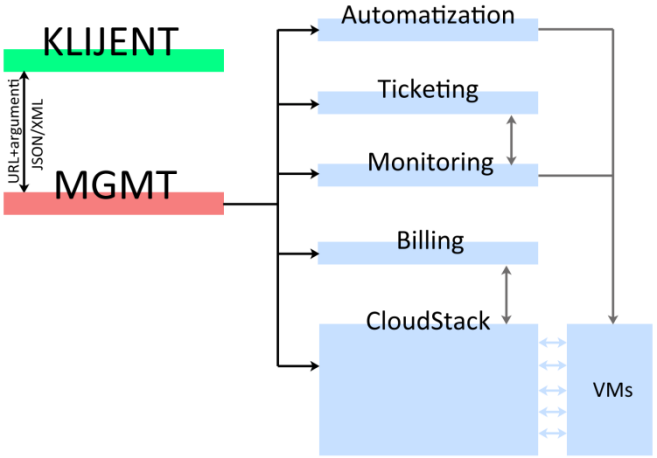
Zbog velikog broja modula različitih namena potrebnih za funkcionisanje sistema za upravljanje klaudom kako bi se podržao veliki broj korisnika, nepraktično je da korisnik pristupa direktno svakom modulu. Ovakav pristup je nepraktičan i zbog bezbednosnog aspekta, jer bi korisnička komunikacija sa sistemom i komunikacija između sistemskih modula koristile iste kanale komunikacije, pa bi bilo teško odbraniti se od eventualnih napada na sam servis. Zbog toga, najpraktičnije bi bilo da korisnik pristupa jednom modulu, koji će sa ostalim modulima kominicirati po potrebi. U slučaju multiplikacije modula iste namene, centralni modul će komunicirati sa najmanje opterećenim modulom tražene namene. U slučaju otkaza jednog modula, sistem balansiranja opterećenja će funkcionalnost prebaciti na drugi, tako da postoji minimalna šansa da korisnički zahtev ostane neizvršen. U najgorem slučaju, korisnik bi bio obavešten o grešci, ali nikada ne bi ostao bez odgovora.

Komponenta sa kojom će korisnik imati najviše kontakta i kroz koju će zapravo pristupati i pokretati funkcije unutar servisa je korisnički interfejs. Od suštinskog značaja je da korisnik u svakom trenutku veruje da sve vreme upravlja istom klaud platformom, odnosno da korisničko iskustvo bude nezavisno od platforme kojom zapravo upravlja.

A. Menadžment modul

Korisnički interfejs prosleđuje korisničke akcije centralnom menadžment modulu putem API zahteva. Korisnik kroz korisnički interfejs može da upravlja zakupljenim resursima, zakupi nove resurse, obriše postojeće. Menadžment modul se nalazi iza balansera opterećenjem, tako da je uvek dostupan korisniku. Zavisno od vrste balansera opterećenja i taktike balansiranja, menadžment modul može slati zahteve balanserima opterećenja određenog modula, koji će zahteve proslediti samom modulu, ili može čuvati adresu trenutno najmanje opterećenog modula tražene namene i komunicirati direktno sa njim. U drugom slučaju bi balanseri opterećenja u slučaju povećanja opterećenja određene instance morali da obaveste menadžment

modul o promeni IP adrese modula određene namene i dostave IP adresu nove instance iste namene. Sa stanovišta fizičke arhitekture, ovaj sistem je visoko dostupan, može da sadrži fizičke ili softverske balansere opterećenja, a suštinski se izvršava na klaud instancama kao bilo koja druga multiservisna ili mikroservisna aplikacija.



Sl. 4. Multiservisna arhitektura za upravljanje klaud resursima

Na Sl. 4 su prikazani osnovni moduli potrebni za funkcionisanje servisa za upravljanje klaud resursima. Neki od modula se koriste za direktnu interakciju sa virtuelnim mašinama, neki se koriste samo za praćenje rada mašina, dok neki nemaju kontakt sa samim mašinama, ali na osnovu informacija sa drugih modula vrše funkcije za koje su namenjeni. Broj i funkcije modula nisu ograničeni i moduli se mogu dodavati, brisati, multiplicirati ili menjati dok god poštuju arhitekturu i biznis logiku koju je potrebno ispratiti.

B. Modul za automatizaciju procesa upravljanja

Ovaj modul sliži da kreiranu instancu sa predefinisanim podešavanjima prilagodi korisničkim zahtevima, da podesi sistemske parametre, instalira potrebne softverske pakete i pripremi instancu za dalji rad. Kada modul za automatizaciju završi sa podešavanjem instance, instanca ima podešene pristupne parametre, monitoring parametre, podignuti su bezbednosni protokoli i ima sve potrebne pakete za instalaciju korisničkog softvera.

C. Modul za obradu tiketa i logova

Svrha ovog modula je da vodi evidenciju o prijavljenim greškama ili zahtevima kreiranim od strane sistema ili korisnika. Tiket može biti kreiran od strane sistema ukoliko dođe do greške za čije otklanjanje je potrebna ljudska intervencija, ili od strane korisnika, ukoliko je u pitanju neki korisnički zahtev ili prijava greške direktno. Ovaj modul takođe sadrži i bazu znanja, pa ukoliko korisnici imaju pitanje koje je neko već pitao, mogu da pronađu odgovor ili ih operater može uputiti na link sa objašnjenjem ili procedurom za rešavanje njihovog problema.

U našem sistemu ovaj modul je takođe čuvao i sistemske logove, ali je zbog velike količine podataka sistem za logovanje praktičnije izdvojiti u poseban servis. Logovi se čuvaju po kategorijama i tagovima, radi kasnije lakše pretrage i sortiranja.

D. Monitoring modul

Isporukom virtuelne mašine ne prestaje isporuka servisa korisniku. Korisnik želi da prati svoju mašinu, da analizira njen rad i na osnovu toga donosi odluke. Da bi se korisnicima što više olakšalo upravljanje, pruža im se funkcionalnost praćenja instanciranih virtuelnih mašina. Ovaj modul takođe služi i za praćenje dostupnosti i iskorišćenja klaud resursa [6], kao i za automatsku prijavu tiketa i logova ukoliko dođe do neke greške. Podaci iz monitoring sistema mogu da se iskoriste i za izračunavanje dostupnosti određene klaud platforme i dalje ocenjivanje i rangiranje klaud platformi na osnovu tih rezultata ukoliko postoji potreba za tim.

Monitoring modul može imati značajnu ulogu u automatskom sklairanju korisničkih sistema i multipliciranju korisničkih servera.

E. Modul za naplatu

Ovaj modul služi za monetizaciju usluga koje se nude. Načini naplate mogu da budu različiti i zavise od biznis logike. Sistem na kome smo radili je podržavao pripejd i postpejd naplatu na mesečnom nivou po virtuelnoj mašini. Sistem koji je opisan u ovom radu dodatno podržava pripejd uplatu sredstava na korisnički račun i naplatu resursa po potrošnji dok god postoje sredstva na računu, zakup određenih resursa i njihovu potrošnju dok god postoje zakupljeni resursi na računu, kao i akcijske ponude resursa, različite cenovnike za različite vrste korisnika, rangiranje korisnika po potrošnji. Za razliku od sistema na čijem razvoju sam radio, sistem koji opisujem u ovom radu može da podrži više lokalnih klaud platformi, ali i klaud platforme postojećih klaud provajdera. To bi značilo da prilikom naplate klaud usluga u cenu mora biti uračunata cena zakupa klaud usluga od drugog provajdera.

Jedno od rešenja je da se modul za naplatu podeli na modul koji čuva informacije o korisničkim finansijama i transakcijama i dodatne module

naplate zavisno od vrste plaćanja, modul za pripejd, modul za postpejd, modul za plaćanje po potrošnji, modul za eksternu platformu. Na ovaj način bi sistem imao podršku za veliki broj različitih načina korišćenja i naplate usluga. Svaka zakupljena usluga imala bi samo jedan aktivan način naplate, odnosno referenca na uslugu bi se u svakom trenutku nalazila samo u jednom modulu koji brine o naplati. Takođe bi promena načina naplate korišćenja virtuelne mašine bila vrlo lako izvodljiva, referenca na virtuelnu mašinu bi se samo prebacila iz jednog modula u drugi i naplata usluge bi odmah mogla da se obavlja na drugi način.

F. Modul za upravljanke klaud platformom

Na Sl. 4 je prikazan Klaudstek modul, koji predstavlja klaud platformu. Klaudstek je uzet za primer jer sam ga koristio kao platformu prilikom razvoja sistema, ali kao što je već rečeno, njegovo mesto može da zauzme bilo koja druga ili više klaud platformi. Da bi se ovo omogućilo, između klaud platforme i ostatka sistema bi morao da postoji međumodul, koji bi uniformne zahteve sistema prilagođavali određenoj klaud platformi. Pod pretpostavkom da svaka klaud platforma ima drugačije API funkcije, broj upravljačkih modula bi bio jednak broju klaud platformi koje želimo u sistemu.

U slučaju klaud platformi postojećih klaud provajdera, modul za upravljanje bi morao da ima dodatne funkcije za proveru cene resursa, koje bi pozivao modul za naplatu.

G. Modul za upravljanje korisnicima

Svrha modula je čuvanje podataka o korisnicima, korisničkim grupama, privilegijama, autentifikacija korisnika, kreiranje korisničkih tokena.

U početku je bio deo menadžment modula, ali je izdvojen u poseban modul zbog uvođenja mikroservisne arhitekture, kompleksnije kontrole pristupa i izmenjene biznis logike koja bi zahtevala složenije korisničke privilegije i korisničke grupe.

VII. PROCESI UPRAVLJANJA KLAUDOM

U daljem tekstu biće objašnjeni primeri korisničih akcija kroz procese koji se izvršavaju kako bi se korisnička želja sprovela u delo.

A. Kreiranje nove virtuelne mašine

Odabir potrebnih resursa za kreiranje nove virtuelne mašine moguće je postići na više načina, ali najpraktičniji je da se unapred definišu paketi resursa koji mogu da se zakupe, tako da se od korisnika zahteva samo jednostavna akcija koja će aktivirati već predefinisani zahtev ka menadžment modulu. U slučaju da je korisniku potrebna drugačija struktura resursa, onda

mu treba omogućiti da sam odabere resurse po želji. Vrlo je bitno da korisnički interfejs menadžment modulu prosledi sve parametre potrebne za izvršenje odabrane akcije.

Nakon što menadžment modul dobije zahtev za kreiranje virtuelne mašine, dobijeni zahtev se parsira i generišu se zahtevi koji će biti prosleđeni drugim modulima. Prvo se sa finansijskim modulom proverava da li korisnik na računu ima odobrena sredstva za odabranu uslugu. Ukoliko je to slučaj, zahtev za kreiranje mašine se prosleđuje modulu za upravljanje klaud platformom. Modul za upravljanje klaudom primljeni zahtev prilagođava ovom slučaju Klaudstek API-u. Zahtev za kreiranje mašine na Klaudsteku ima tri obavezna parametra, dok ostali parametri mogu, ali ne moraju da se navedu, zavisno od arhitekture klaud platforme. Obavezni parametri za kreiranje mašine na Klaudstek platformi su serviceofferingid, templateid i zoneid.

Serviceofferingid predstavlja unapred određen skup fizičkih resursa koji će biti dodeljeni virtuelnoj mašini. Moguće je da argumenti nekih resursa u skupu resursa budu samo deklarisani, bez dodeljene vrednosti. U tom slučaju je potrebno dodatno navesti sve potrebne vrednosti za kreiranje virtuelne mašine u vidu argumenata unutar API zahteva.

Templateid predstavlja unapred definisani templejt[2] ili ISO imidž[3] iz kog će se instalirati operativni sistem prilikom prvog pokretanja mašine. U slučaju templejta, nova virtuelna mašina ne instalira operativni sistem i servise, već je moguće dobiti potpuno funkcionalnu mašinu za par minuta, umesto jednog ili dva sata koliko traje instalacija operativnog sistema iz ISO fajla.

Zoneid predstavlja zonu unutar Klaudstek infrastrukture u kojoj se kreira nova mašina.

Primer API poziva:

*http://server.com:8080/client/api?***command**=*deployVirtualMachine*&**serviceofferingid**=*d4023a86*&**zoneid**=*f05f877b*&**templateid**=*2e943696*&**networkids**=*0070f932*&**details[0].cpuSpeed**=*2000*&**details[0].cpuNumber**=*2*&**details[0].memory**=*4096*&**hypervisor**=*XenServer*&**name**=*VM1*&**displayname**=*VM1* &**domainId**=*aee52bd2*&**account**=*user1@server.com*&**response**=*json*& **apiKey**=*pWlkWpakr* &**signature**=*XO4GQHKz3rmvRCU7jkgwlLF20e4=*

[2] Templejt (template) u ovom slučaju je kopija hard diska virtuelne mašine na kojoj su prethodno instalirani operativni sistem i servisi, a koji se dodeljuje novoj virtuelnoj mašini.

[3] ISO imidž (image) je vrsta fajla koji predstavlja kopiju CD diska sa očuvanom strukturom sektora i fajl sistemom, na kome se može nalaziti operativni sistem.

Parametri koje primećujemo su command, serviceofferingid, zoneid, templateid, networkids, details[0].cpuSpeed, details[0].cpuNumber, details[0].memory, hypervisor, name, displayname, domainId, response, apiKey, signature. Od toga su Serviceoffering, zoneid, templateid, command, response, apiKey i signature već objašnjeni u dosadašnjem tekstu.

Razlog za ovoliki broj parametara je što odabrani serviceoffering nema definisane parametre za mrežu, broj procesora, brzinu procesora, količinu memorije, administratorski domen, nalog i hipervizora. Dakle, parametri su unutar odabranog serviceoffering-a samo deklarisani, ali nemaju dodeljene vrednosti, pa je potrebno proslediti dodatne parametre unutar API zahteva. Na osnovu navedenog zahteva, biće kreirana mašina u administratorskom root domenu, na nalogu user1, sa imenom VM1 i adresom iz definisanog opsega, sa dva procesora brzine 2000MHz po jezgru i 4GB rama. U ovom zahtevu veličina hard diska nije definisana, jer je navedeni templateid unapred kreiran templejt sa predefinisanom veličinom hard diska i instaliranim operativnim sistemom. Da to nije slučaj, bilo bi potrebno da se navede i veličina hard diska kroz parametar diskofferingid. Iako se koristi predefinisani templejt u kome je definisana veličina hard diska, moguće je u API zahtevu proslediti parametar rootdisksize kojim se definiše nova veličina diska. Takođe je moguće unapred odrediti IP adresu virtuelne mašine kroz parametar ipaddress.

Pošto je deployVirtualMachine asinhrona[4] komanda, u odgovoru koji Klaudstek vraća kao objekat, primećuje se argument jobID.

JSON odgovor:

```
{ "deployvirtualmachineresponse" : {
   "id":"24f30aa4-7df1-4b96-b285-19eb660b56a3",
   "jobid":"2d623af9-643c-4ae9-a3b9-e939e3599f3f"
}}
```

Klaudstek ima implementiranu komandu za proveru statusa poslova queryAsyncJobResult, koja kao argument prosleđuje jobID, a kao odgovor dobija informaciju o statusu izvršavanja komande. Zahtev koji prosleđuje izgledao bi ovako:

http://server.com:8080/client/api?command=queryAsyncJobResult&jobid=2d623af9-643c-4ae9-a3b9-e939e3599f3f

Kao odgovor na ovakav zahtev, Klaudstek bi vratio informacije o konkretnom poslu, ko ga je kreirao, kada, koja komanda bi trebalo da se izvrši, nad kojim resursom, kao i status samog posla predstavljen

[4] Koncept asinhronih komandi postoji zbog komandi čije izvršavanje može da potraje, što bi uzrokovalo "time out", odnosno istek maksimalnog vremena u kome CloudStack treba da vrati odgovor. Umesto toga CloudStack komandu izvršava u zasebnoj niti (thread), dok glavna nit kao odgovor vraća jobID koji se koristi za proveru statusa komandi koristeći poisebne API pozive.

argumentom jobstatus. Ovako izgleda dgovor Klaudsteka za posao koji još čeka na izvršenje, u JSON obliku:

```
{ "queryasyncjobresultresponse" : {
   "cmd":"org.apache.cloudstack.api.command.user.vm.DeployVMCmd",
   "jobstatus":0,
   "jobinstancetype":"VirtualMachine",
   "jobid":"2d623af9-643c-4ae9-a3b9-e939e3599f3f"
   ...
}}
```

Ukoliko posao čeka na izvršenje, njegov status je 0, ukoliko posao nije uspešno završen, status posla je 2, a ukoliko je uspešno završen, status posla je 1. Osim informacija o poslu, u slučaju da je on uspešno završen, Klaudstek vraća ceo objekat nad kojim je komanda izvršena, odnosno objekat virtuelne mašine u ovom slučaju.

Ukoliko je posao kreiranja virtuelne mašine uspešno završen, JSON oblik bitnih informacija iz odgovora će izgledati ovako:

```
{ "queryasyncjobresultresponse" : {
      "cmd":"....user.vm.DeployVMCmd",
      "jobstatus":1,
      "jobresult":{
         "virtualmachine":{
            "id":"4d9cd3b9-8596-4fd0-b778-6f31be91a091",
            "name":"VM-1",
            "state":"Running",
            "templatename":"CENTOS-6.7-x64-CLI-20GB",
      }},
      ... //informacije koje nisu bitne za primer
}}
```

Nakon uspešnog kreiranja, status kreirane mašine je Runnung, što se može videti iz odgovora. To znači da je iz perspektive klaud platforme virtuelna mašina spremna za dalju upotrebu. Menadžment modul referencu na virtuelnu mašinu prosleđuje modulu za automatizaciju, koji priprema virtuelnu mašinu za upotrebu, instalirajući i podešavajući potrebne softvere. Nakon što menadžment modul dobije odgovor od modula za automatizaciju, instanca se unosi u monitoring, čime počinje njeno praćenje i u odgovarajući modul za naplatu, čime počinje naplata te usluge. Nakon toga se korisnik obaveštava o uspešnoj isporuci usluge, čime se proces kreiranja završava. U toku procesa, menadžment modul je zadužen za upis logova u sistem.

B. Izmena fizičkih resursa virtuelne mašine

Ako korisnik žečo da izmeni resurse virtuelne mašine, odabraće mašinu i pokrenuti akciju skaliranja. Proces skaliranja je vrlo sličan procesu kreiranja. Zahtev o izmeni se prosleđuje menadžment modulu, koji sa finansijskim modulom proverava da li korisnik ima dovoljno sredstava za traženu izmenu. Zatim se zahtev za izmenu prosleđuje modulu za upravljanje klaud platformom, koji zahtev prilagođava klaud platformi.

Komanda koja se prosleđuje Klaudsteku je scaleVirtualMachine, koja ima dva obavezna parametra: id koji predstavlja d virtuelne mašine i serviceofferingid koji predstavlja unapred definisan skup resursa. Pošto navedeni skup resursa ima samo deklarisane promenjive bez dodeljenih vrednosti, vrednosti koje korisnik navede biće prosleđene kao nove vrednosti, a sve ostale će biti ostavljene kakve jesu.

Ukoliko korisnik odluči da mu je potrebna virtuelna mašina sa tri procesorska jezgra brzine 3GHz i 2048MB rama, zahtev koji će biti prosleđen CloudStack-u će izgledati ovako:

*http:/server.com:8080/client/api?***command***=scaleVirtualMachine&***serviceofferingid***=d4023a86&***response***=json&***id***=4d9cd3b9&***details[0].cpuNumber***=3&***details[0].memory***=2048&***details[0].cpuSpeed***=3000&***apiKey***=pWlkWwak&***signature***=E4bBP3m6UZw3mU7w2ABErz6PgR4=*

Iz ovog zahteva se može videti da se osim obaveznih argumenata među argumentima nalaze i tri dodatna argumenta cpuNumber, memory i cpuSpeed. Zajedničko za sva tri je što su podargumenti prvog člana niza argumenata „details“. U ovom nizu se definišu svi dodatni argumenti koji nemaju vrednost u skupu resursa koji je naveden serviceofferingid argumentom.

Ova komanda je asinhrona i kao odgovor vraća jobid. Modul za upravljanje klaud platformom će ispitivati status posla korišćenjem ranije opisane procedure sve dok statusa posla ne postane 1 ili 2. Ukoliko postane 1, odgovor koji se dobija od Klaudsteka će sadržati objekat izmenjene virtuelne mašine sličan uspešnom odgovoru iz prethodnog primera, dok će u drugom slučaju biti opisan razlog zbog kojeg skaliranje nije bilo uspešno.

Upisivanjem logova i obaveštavanjem korisnika o ishodu akcije, završava se proces izmene fizičkih resursa.

C. Pokretanje i zaustavljanje virtuelne mašine

Pokretanje i zaustavljanje virtuelne mašine je omogućeno kroz korisnički grafički interfejs tako da korisnik jednim klikom može da zaustavi virtuelnu mašinu, da je pokrene ili rebutuje. Kada takav zahtev stigne do menadžment modula, biva prosleđen modulu za upravljanje klaud platformom, koji ga u

slučaju Klaudsteka prevodi u jednu od tri komande, startVirtualMachine, stopVirtualMachine ili rebootVirtualMachine. Svaka od ove tri komande je asinhrona, te vraća jobid. Zajedničko za sve tri komande je da im je id virtuelne mašine obavezan parametar i nakon završenog posla vraćaju objekat virtuelne mašine kao odgovor na queryAsyncJobResult.

Komanda startVirtualMachine ima i opcione parametre deploymentplanner i hostid, koji služe za balansiranje opterećenjem na nivou organizacionih jedinica u Klaudsteku. Deploymentplanner prosleđuje odabrani algoritam CloudStack-u, kako bi se na osnovu njega automatski odredilo gde će virtuelna mašina biti pokrenuta. Hostid daje izvršiocu zahteva mogućnost da odredi na kom fizičkom hostu će mašina biti pokrenuta.

Komanda stopVirtualMachine ima samo jedan opcioni parametar - forced, koji ako ima vrednost true forsira stopiranje virtuelne mašine iako je na njoj u toku izvršavanje nekog procesa.

D. Brisanje virtuelne mašine

Prilikom brisanja mašine korisnika treba upozoriti da je brisanje nepovratna akcija. Stoga bi trebalo uvesti dodatni korak autentifikacije ili potvrde akcije kako korisnik ne bi greškom obrisao mašinu.

Ako se korisnik ipak odluči na brisanje mašine, komanda koja bi bila prosleđena Klaudstek platformi je destroyVirtualMachine. Obavezan parametar je id virtuelne mašine, opcioni parametar je expunge. Ukoliko je vrednost expunge parametra true, Klaudstek će prilikom brisanja mašine obrisati i njen hard disk, sa sve podacima. Ukoliko expunge nije naveden, podrazumevana vrednost je false i Klaudstek će čekati istek podrazumevanog intervala za brisanje (default expunge interval), nakon čega će hard diskovi svih virtuelnih mašina označenih za brisanje biti nepovratno obrisani. Kao i u prethodnim slučajevima, Klaudstek vraća jobid kao odgovor na zahtev.
API zahtev za brisanje virtuelne mašine izgleda ovako:
http://server.com:8080/client/api?**command**=*destroyVirtualMachine*&**apiKey**=*pWlkWwakr9tj18PUFRlZPquVIKzxDeZim1IA*&**signature**=*ag32FHO89lFSl4*
=

E. Ostale komande za upravljanje Klaudstek platformom

Klaudstek API, ima mnoštvo komandi za upravljanje klaud sistemom. Većina komandi ima dodatne argmente koji određuju njihovo ponašanje. Odgovor koji vraćaju je i pored toga uniforman, što omogućava relativno laku integraciju platforme u već postojeći sistem. Klaudstek je uzet kao primer klaud platforme jer može da koristi unapred definisane templejte za brzo kreiranje virtuelnih mašina, što se u praksi pokazalo kao odlično rešenje

za kreiranje virtuelnih mašina sa operativnim sistemom za par minuta. Ovo je značajno skraćenje vremena i ima ogroman uticaj na korisničko iskustvo.

Takođe, iako Klaudstek ima implementirano automatsko raspoređivanje mašina po hostovima klaud sistema, moguće je osmisliti i implementirati sopstvena pravila koristeći postojeće komande. Ova funkcionalnost se može primeniti za balansiranje opterećenja, povećanje dostupnosti korisničkih sistema unutar distribuiranog klaud okruženja ili smanjenje kašnjenja odgovora. Tako se virtuelna mašina može migrirati unutar klaud sistema sa jednog hosta na drugi koji se fizički nalazi na drugoj lokaciji, ukoliko to ima uticaj na bolje korisničko iskustvo.

Za ovakve funkcionalnosti je potrebno izvesno znanje o stanju sistema, a informacije o sistemu ili o pojedinačnim resursima se mogu dobiti koristeći list komande. Ako bismo želeli da dobijemo informacije o virtuelnim mašinama, koristili bismo komandu listVirtualMachines. Ukoliko bismo želeli da filtriramo virtuelne mašine po nekom osnovu, koristili bismo neki dadatni argument unutar API poziva po kome će se mašine filtrirati. Konkretno, u slučaju da nam je potrebna virtuelna mašina sa određenim ID-em, koristili bismo argument id:

*http://server.com:8080/client/api?***command***=listVirtualMachines&***response***=json&***id***=e5c38&***domainId***=aeed2&***apiKey***=pW9tj18&***signature***=30afa=*

Ova komanda kao odgovor vraća objekat virtuelne mašine sa svim relevantnim informacijama o mašini. Te informacije se mogu iskoristiti za izračunavanje statistika sistema ili same virtuelne mašine, za dobijanje dodatnih informacija, itd.

VIII. ZAKLJUČAK

Iako u jednom trenutku potisnut, zbog jednostavnosti razvoja kompleksnih skalabinlih arhitektura REST je doživeo ponovnu popularizaciju i postao gotovo standard za izgradnju distribuiranih veb aplikacija.

Predstavljena arhitektura koristi REST baš iz tog razloga. Ona je nastala kao rezultat dvogodišnjeg razvoja komercijalne platforme za upravljanje klaud resursima u kombinaciji sa ličnom željom da se prevaziđu konceptualne greške koje su stvarale praktične probleme i ograničenja u pružanju usluga tokom i nakon razvoja pomenute platforme.

Primena ovakvog sistema bi mogla biti komercijalna usluga, koja korisniku pruža jednostavan interfejs, koji u pozadini objedinjuje sve aspekte upravljanja klaudom. Takođe bi primena mogla biti unutar organizacija, instituta ili univerziteta. Klaud resursi na nivou univerziteta, organizacije ili države bi mogli biti ujedinjeni pod ovim sistemom, a onda bi se određenim grupama odobravala sredstva koja se mogu trošiti unutar sistema. Na taj

način bi višak resursa mogao da se monetizuje izdavanjem resursa naučnim ili komercijalnim projektima sa određenim trajanjem, a dobijena sredstva bi se dalje ulagala u razvoj sistema ili infrastrukture.

LITERATURA

[1] Hyuck Han, Shingyu Kim, Hyungsoo Jung, Yeom, H.Y., Changho Yoon, Jongwon Park, Yongwoo Lee (2009). A RESTful Approach to the Management of Cloud Infrastructure; Cloud Computing, 2009. CLOUD '09. IEEE International Conference, IEEE, DOI: 10.1109/CLOUD.2009.68

[2] Roy Thomas Fielding (2000). Architectural Styles and the Design of Network-based Software Architectures; Dissertation UNIVERSITY OF CALIFORNIA, IRVINE (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

[3] Tim O'Reilly (2009). What is web 2.0? : design patterns and business models for the next generation of software; Sebastopol, CA : O'Reilly Media, Inc. ISBN: 9781449391706 1449391702 9781449391072 1449391079

[4] CloudStack API Reference Documentation (http://docs.cloudstack.apache.org/en/latest/dev.html)

[5] Kirschnick, J., Calero, J. M. A., Wilcock, L., & Edwards, N. (2010). Toward an architecture for the automated provisioning of cloud services. IEEE Communications Magazine, 48(12), 124-131.

[6] De Chaves, S. A., Uriarte, R. B., & Westphall, C. B. (2011). Toward an architecture for monitoring private clouds. IEEE Communications Magazine, 49(12), 130-137.

ABSTRACT

The aim of this paper is the presentation of the architecture of the cloud infrastructure management system that uses REST principles and REST API for a communication between distributed system modules. It explains the advantages and disadvantages of the presented systems and communication with the cloud platform in the example of CloudStack. Examples of application in the real world are given. The idea of this architecture is a result of the personal challenge to overcome conceptual mistakes which were encountered in the service offering during and after two years of development of a commercial cloud management platform that first author was working on.

SYSTEM FOR CLOUD MANAGEMENT USING THE REST

Dušan Kilibarda, dr Dušan Vujošević