UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matjaž Poljanšek

Analiza modela platforme kot storitve in razvoj aplikacije v oblaku na platformi Google App Engine

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01772/2011
Datum: 01.09.2011

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

Kandidat: **MATJAŽ POLJANŠEK**

Naslov: **ANALIZA MODELA PLATFORME KOT STORITVE IN RAZVOJ APLIKACIJE V OBLAKU NA PLATFORMI GOOGLE APP ENGINE**

ANALYSIS OF PLATFORM-AS-A-SERVICE MODEL AND CLOUD APPLICATION DEVELOPMENT ON GOOGLE APP ENGINE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite model platforme kot storitve (Platform as a service, PaaS). Predstavite glavne koncepte in prednosti računalništva v oblaku ter primerjajte PaaS z modelom infrastrukture kot storitve (Infrastructure as a service, IaaS) in programske opreme kot storitve (Software as a service, SaaS). Podrobno proučite PaaS, njegove karakteristike, prednosti in slabosti, opravite primerjavo s tradicionalnim modelom razvoja aplikacij. Razvijte aplikacijo v oblaku v Javi ter opišite izzive pri razvoju na platformi Google App Engine.

Mentor:

prof. dr. Matjaž B. Jurič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Matjaž Poljanšek**,

z vpisno številko **63040132**,

sem avtor diplomskega dela z naslovom:

Analiza modela platforme kot storitve in razvoj aplikacije v oblaku na platformi Google App Engine

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  prof. dr. Matjaža Branka Juriča

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 12.12.2011 Podpis avtorja:

Zahvala

Na prvem mestu se iskreno zahvaljujem svojemu mentorju, prof. dr. Matjažu Branku Juriču za vso pomoč in nasvete pri izdelavi diplomske naloge.

Posebna zahvala gre mojim staršem, mami Faniki in očetu Bogdanu, ki sta me tekom študija vedno podpirala, in ki sta mi in vedno bosta, s svojo ljubeznijo, skrbnostjo in delavnostjo največja vzornika v življenju.

Največjo zahvalo pa dolgujem moji Nataši. Njena ljubezen in podpora sta mi vlivali upanje tudi v najtežjih trenutkih. Brez nje bi bila diplomska naloga še vedno nekje »v oblakih«.

Staršem

Kazalo

Seznam uporabljenih kratic

aPaaS	Application Platform As A Service (aplikacijska platforma kot storitev)
API	Application Programming Interface (aplikacijski programski vmesnik)
ASP	Application Service Provider (model ponudnika aplikacijskih storitev, predhodnik modela PaaS)
B2B	Business-To-Business (elektronsko poslovanje med poslovnimi sistemi)
ESB	Enterprise Service Bus (nabor principov in metodologij za povezovanje aplikacij v storitveno usmerjeni arhitekturi)
HTML	HyperText Markup Language (standardni označevalni jezik za izdelavo spletnih strani)
IaaS	Infrastructure As A Service (infrastruktura kot storitev)
IDE	Integrated Development Environment (integrirano razvojno okolje)
iPaaS	Integration Platform As A Service (povezovalna platforma kot storitev)
JSP	JavaServer Pages (tehnologija za izdelavo dinamičnih spletnih strani)
JVM	Java Virtual Machine (Java navidezna naprava)
LDAP	LightWeight Directory Access Protocol (mehanizem za avtentikacijo)
NIST	The National Institute of Standards and Technology (ameriški Nacionalni inštitut za standarde in tehnologijo)
OCCI	Open Cloud Computing Interface (odprt vmesnikAPI za računalništvo v oblaku)
PaaS	Platform As A Service (platforma kot storitev)
PB	Podatkovna baza

REST	Representational State Transfer (množica arhitekturnih principov za razvoj spletnih storitev)
SaaS	Software As A Service (programska oprema kot storitev)
SAML	Security Assertion Markup Language (implementacija povezane identifikacije)
SDK	Software Development Kit (nabor orodij za razvoj programske opreme)
SLA	Service Level Agreement (dogovor o nivoju storitev)
SOA	Service-Oriented Architecture (storitveno usmerjena arhitektura)
SOAP	Simple Object Access Protocol (standard za razvoj spletnih storitev, ki temelji na XML)
SQL	Structured Query Language (poizvedovalni jezik v relacijskih podatkovnih bazah)
SSL	Secure Sockets Layer (kriptografski protokol za varno komunikacijo na spletu)
URL	Uniform Resource Locator (enoličen naslov spletne strani)
VM	Virtual Machine (navidezna naprava)
WAR	Web Application Archive (struktura virov za spletne aplikacije)
XML	Extensible Markup Language (format podatkov za izmenjavo strukturiranih dokumentov v spletu)

Povzetek

V diplomskem delu smo obravnavali model platforme kot storitve (*Platform as a Service, PaaS*). Gre za enega izmed treh modelov storitev računalništva v oblaku, ki je namenjen razvoju in izvajanju aplikacij v oblaku. V uvodnem poglavju so predstavljeni glavni koncepti in prednosti računalništva v oblaku ter opis in primerjava med vsemi tremi modeli, kamor poleg PaaS spadata še model infrastrukture kot storitve (*Infrastructure as a Service, IaaS*) ter model programske opreme kot storitve (*Software as a Service, SaaS*). Glavni del naloge zajema podroben opis modela PaaS, njegove karakteristike, prednosti in slabosti, primerjavo s tradicionalnim modelom razvoja aplikacij ter trende na tem področju. Opisan je tudi večnajemniški model, kot temelj aplikacij v oblaku ter opis in primerjava med štirimi pristopi pri njegovi implementaciji. Posebno poglavje je namenjeno analizi trga ponudnikov rešitev PaaS ter opisu in primerjavi štirih izbranih ponudnikov in njihovih platform: Microsoft Windows Azure, Salesforce.com Force.com, Amazon Web Services Elastic Beanstalk in Google App Engine. Na slednji smo razvili in postavili tudi primer spletne aplikacije v Javi. V zaključnem poglavju smo opisali izzive, s katerimi smo se srečali pri razvoju spletne aplikacije na platformi Google App Engine ter podali naše videnje prihodnosti modela PaaS.

Ključne besede:

Računalništvo v oblaku, PaaS, večnajemniški model, Google App Engine

Abstract

In the thesis we discuss the Platform as a Service (PaaS) model. It is one of the three service delivery models of cloud computing, intended for developing and running applications in the cloud. The introductory chapter presents main concepts and advantages of cloud computing, as well as a description and comparison between all three models, including Infrastructure as a Service (IaaS) and Software as a Service (SaaS). The main part of the thesis includes a detailed description of the PaaS model, its characteristics, advantages and disadvantages, the comparison with a traditional model of application development and the current trends in this field. There is also the description of the multitenancy model, an essential concept of cloud applications, and the description and comparison between the four approaches of its implementation. A separate chapter features an analysis of the PaaS market and a description and comparison of the chosen four providers and their cloud platforms: Microsoft Windows Azure, Salesforce.com Force.com, Amazon Web Services Elastic Beanstalk and Google App Engine. On the latter, we have developed and deployed an example of a Java web application. In the final chapter we describe the challenges we encountered while developing the web application on the Google App Engine platform and give our opinion on the future of PaaS.

Key words:

Cloud computing, PaaS, multitenancy model, Google App Engine

Uvod

Računalništvo v oblaku (*Cloud Computing*) je pojem, ki je v zadnjih štirih letih povzročil pravo revolucijo tako na tehnološkem kot tudi poslovnem področju in danes bi verjetno težko našli posameznika, ki za to besedno zvezo še ni slišal.

Oblak prinaša nov način uporabe aplikacij. Te so zdaj ponujene v uporabo kot storitev prek spleta. Na poslovnem področju tako prihaja do sprememb v poslovnih modelih, saj se z uporabo storitev v oblaku poslovne organizacije lahko bolj osredotočajo na samo poslovanje.

Računalništvo v oblaku pa prinaša tudi prednosti in nove izzive za razvijalce. Tudi platforma za razvoj aplikacij je na voljo v obliki storitve, in sicer gre za t.i. model platforme kot storitve (*Platform as a Service, PaaS*). Aplikacija za svoje delovanje potrebuje določeno strojno in programsko opremo, ki je lahko zelo draga. S pojavom modela PaaS je večina stroškov in kompleksnost povezana z vrednotenjem, kupovanjem, prilagajanjem in upravljanjem strojne in programske opreme, ki jo aplikacije potrebujejo za svoje delovanje, preložena na ponudnika platforme v oblaku. Posledično to za razvijalce pomeni, da se lahko bolj posvetijo samemu razvoju aplikacij.

Cilj diplomske naloge je podrobno analizirati model PaaS in na podlagi analize podati mnenje o tem, ali utegne v prihodnosti nadomestiti tradicionalni model razvoja aplikacij.

Diplomska naloga je razdeljena na pet poglavij. V prvem poglavju *Računalništvo v oblaku* bomo predstavili definicijo računalništva v oblaku in njegove glavne koncepte. Opisali in primerjali bomo vse tri modele storitev SaaS, PaaS in IaaS ter izpostavili prednosti računalništva v oblaku.

Drugo poglavje *Model platforme kot storitve* je namenjeno podrobni analizi modela PaaS. V njem bomo opisali njegove glavne karakteristike, primerjavo s tradicionalnim modelom razvoja aplikacij ter prednosti in slabosti. Opisali bomo tudi večnajemniški model, ki je zelo pomemben koncept pri razvoju aplikacij v oblaku in predstavili ter primerjali štiri pristope pri njegovi implementaciji. Opisali bomo še aplikacijske programske vmesnike, kot glavni način razvoja aplikacij v oblaku, model zaračunavanja storitev ter trende na področju PaaS.

Tretje poglavje *Primerjava glavnih ponudnikov rešitev PaaS na trgu* bo osredotočeno na analizo obstoječega in prihodnjega stanja trga PaaS ter na opis in primerjavo med štirimi izbranimi ponudniki in njihovimi rešitvami: Microsoft Windows Azure, Salesforce.com Force.com, Amazon Web Services Elastic Beanstalk in Google App Engine.

V četrtem poglavju *Razvoj spletne aplikacije in postavitev na platformi Google App Engine* bomo opisali razvoj in postavitev testne spletne aplikacije v Javi z uporabo platforme Google App Engine.

V zaključnem poglavju bomo opisali ugotovitve in težave, s katerimi smo se srečevali pri razvoju testne aplikacije, ter podali mnenje o prihodnosti modela PaaS.

Poglavje 1

Računalništvo v oblaku

1.1 Kaj je računalništvo v oblaku

Računalništvo v oblaku je pristop, ki kateremkoli uporabniku omogoča, da do aplikacij in storitev dostopa kjerkoli in to s pomočjo katerekoli naprave.

Ključni koncept računalništva v oblaku je, da se računanje izvaja v oblaku. Vsa potrebna strojna in programska oprema ter sami podatki se nahajajo v oblaku in so ponujeni v uporabo prek spleta kot storitev, ki jo nudi ponudnik storitev v oblaku (*Cloud Service Provider*) (slika 1.1). Procesiranje podatkov se tako za razliko od standardnega pristopa ne izvaja več na specifičnih in poznanih strežnikih [25].

Čeprav je računalništvo v oblaku vroča in revolucionarna tema zadnjih štirih let, pa sam koncept sega že v šestdeseta leta 19. stoletja. Že takrat je John McCarthy predvideval, da bo računanje nekega dne organizirano kot javna storitev, podobno kot električna energija [25].

Izraz »oblak« je osnovan na podlagi skice oblaka, ki se je v preteklosti uporabljal za ponazoritev telefonskega omrežja, kasneje pa za ponazoritev interneta v diagramih računalniških omrežij [25].

Oblak zagotavlja abstrakcijo storitev in virov, tako da so podrobnosti njegove tehnološke infrastrukture nevidne uporabnikom in aplikacijam, ki so v interakciji z oblakom. Posledično to pomeni, da končni uporabniki ne potrebujejo več strokovnega znanja o tehnološki infrastrukturi in nadzora nad njo.

Računalništvo v oblaku je pravzaprav nadgradnja sledečih obstoječih tehnologij [25]:

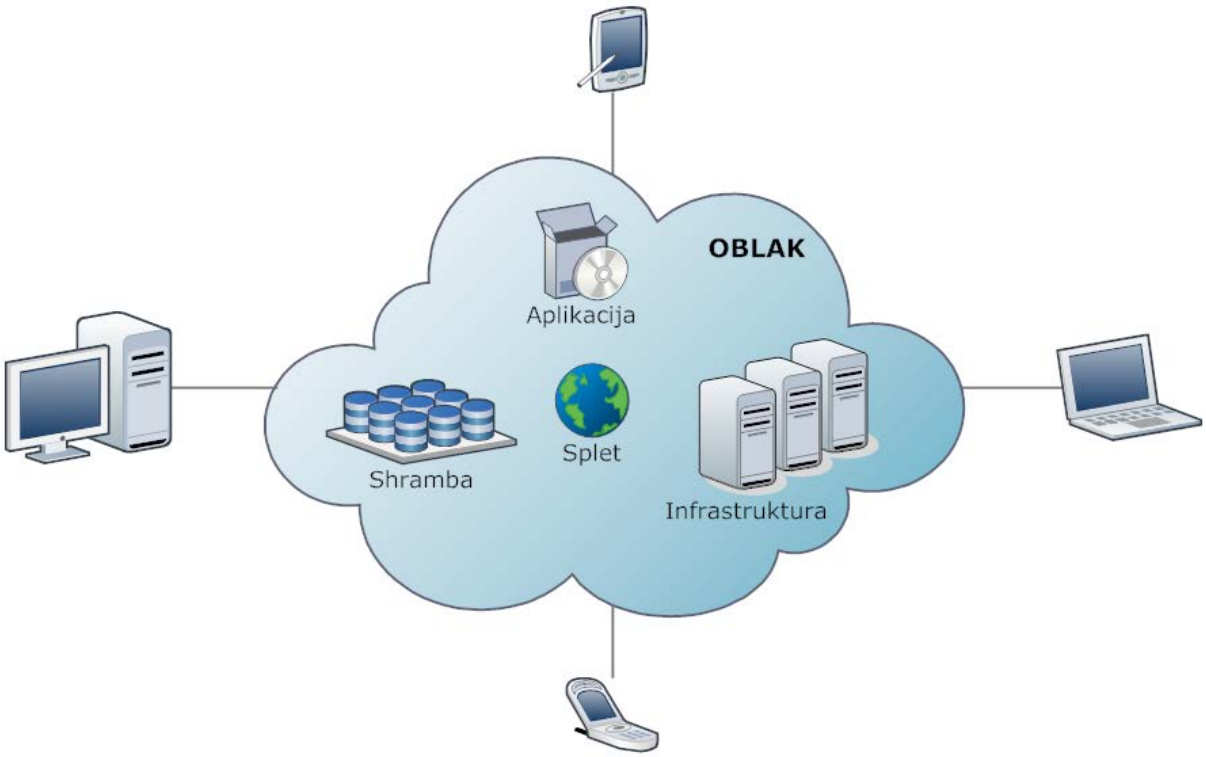
- **Mrežno računalništvo** (*Grid Computing*): gre za skupino računalnikov, ki so povezani v mrežo z namenom skupnega izvajanja zahtevnih nalog.

- **Virtualizacija** (*Virtualization*): gre za kreiranje navideznih naprav (*Virtual Machine, VM*), ki se obnašajo kot pravi, fizični računalniki z operacijskim sistemom. Na enem fizičnem računalniku ali gostitelju se lahko nahaja več navideznih naprav. Programska oprema se na teh navideznih napravah izvaja v neodvisnosti od strojne opreme gostitelja. Prednosti virtualizacije so manjši stroški povezani s strojno opremo, olajšano upravljanje in nadzor, prenosljivost ter lažje obnavljanje v primeru nesreč.

- **Storitveno usmerjena arhitektura** (*Service-Oriented Architecture, SOA*): gre za nabor principov in metodologij za načrtovanje in razvoj programske opreme, kjer so programske komponente storitve, ki lahko med sabo komunicirajo in se ponovno uporabljajo.

- **Avtonomno računalništvo** (*Autonomic Computing*): gre za sposobnost porazdeljenih računalniških sistemov, da se sami odzivajo na spremembe in tako uporabnikom olajšajo upravljanje.

- **Storitveno računalništvo** (*Utility Computing*): gre za uporabo računalniških virov, kot so računanje, shramba in mreža, ki temelji na principu javnih storitev (elektrika, voda, telefon). Namesto kupovanja strojne in programske opreme uporabnik računalniške vire najame, pri čemer plača le toliko, kolikor porabi.



Slika 1.1. Shema računalništva v oblaku

1.2 Uradna definicija računalništva v oblaku

Obstaja veliko definicij računalništva v oblaku, danes pa je v javnosti najbolj široko sprejeta definicija, ki jo podaja ameriški Nacionalni inštitut za standarde in tehnologijo (*The National Institute of Standards and Technology, NIST*) [13]:

»*Računalništvo v oblaku je model, ki preko omrežja in na zahtevo omogoča vseprisoten, priročen dostop do skupnih in nastavljivih računalniških virov (npr. omrežja, strežnikov, shrambe, aplikacij in storitev), in ki ga je mogoče zagotoviti in uvesti z minimalnim naporom upravljanja ali interakcije s ponudnikom storitve. Model oblaka spodbuja razpoložljivost in je sestavljen iz petih glavnih karakteristik, treh modelov storitev in štirih modelov vzpostavitve.«*

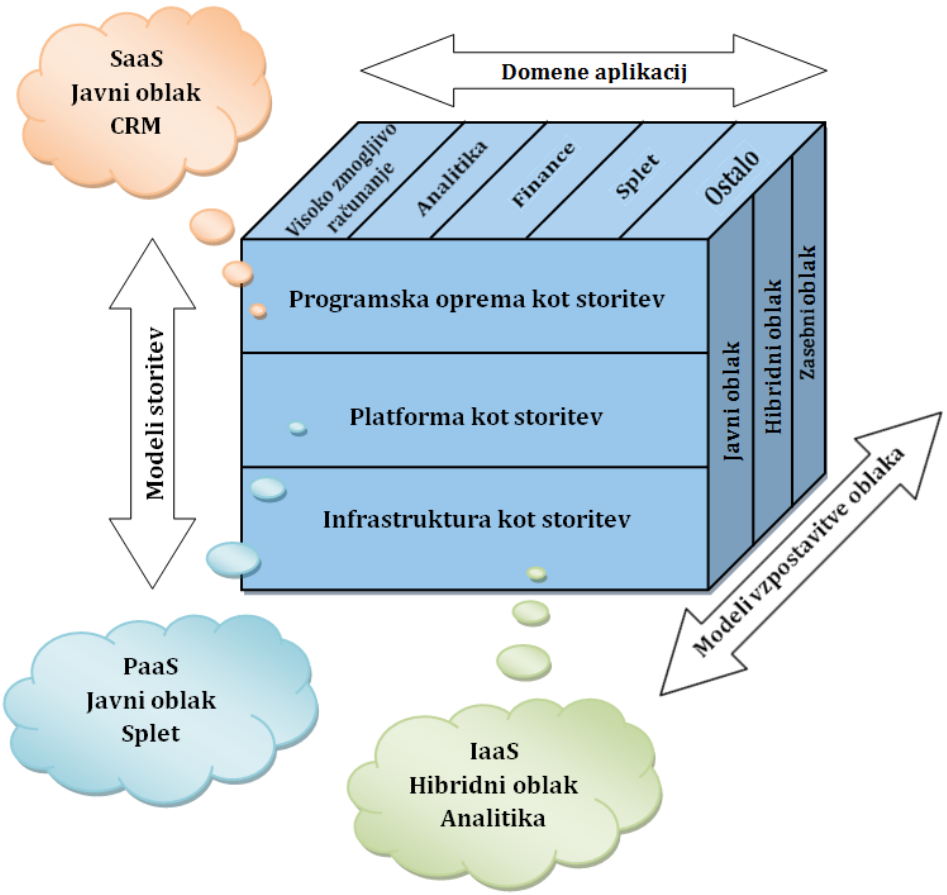
1.3 Glavne karakteristike

NIST podaja pet glavnih karakteristik računalništva v oblaku [13]:

- **Storitev na zahtevo** (*On-demand self-service*): uporabnik lahko računalniške vire, kot so shramba, procesiranje, pomnilnik, pasovna širina in navidezne naprave, zaseda samodejno, brez potrebne interakcije s ponudnikom storitve.
- **Širok dostop do omrežja** (*Broad network access*): računalniški viri so na voljo prek omrežja in dostopni preko različnih odjemalcev (npr. mobilni telefoni, prenosniki in dlančniki).
- **Združevanje virov** (*Resource pooling*): računalniški viri so združeni, da so na voljo večim uporabnikom. Fizični in navidezni viri se dinamično dodeljujejo uporabnikom glede na njihove zahteve, brez da bi ti morali poznati in nadzorovati njihovo lokacijo.
- **Hitra elastičnost** (*Rapid elasticity*): računalniški viri se lahko zasedajo in sproščajo glede na obremenitev. Uporabnik ima občutek, kot da so razpoložljive zmogljivosti na voljo kadarkoli in v neomejenih količinah.
- **Merjenje storitev** (*Measured Service*): sistemi oblakov samodejno nadzorujejo in optimizirajo uporabo virov z uporabo meritev na nivoju abstrakcije, ki ustreza posameznemu tipu storitve (npr. shramba, procesiranje, pasovna širina). S spremljanjem, nadzorovanjem in poročanjem o uporabi virov se zagotavlja transparentnost tako na strani ponudnika kot tudi odjemalca storitve.

1.4 Ogrodje SPI

Splošno sprejeto ogrodje za opis računalništva v oblaku se imenuje SPI [12]. SPI je kratica za vse tri modele storitev, ki sestavljajo računalništvo v oblaku - SaaS, PaaS in IaaS. Ogrodje prikazuje spodnja slika 1.2 [12], več o posameznih komponentah pa sledi v naslednjih podpoglavjih.



Slika 1.2. Ogrodje SPI

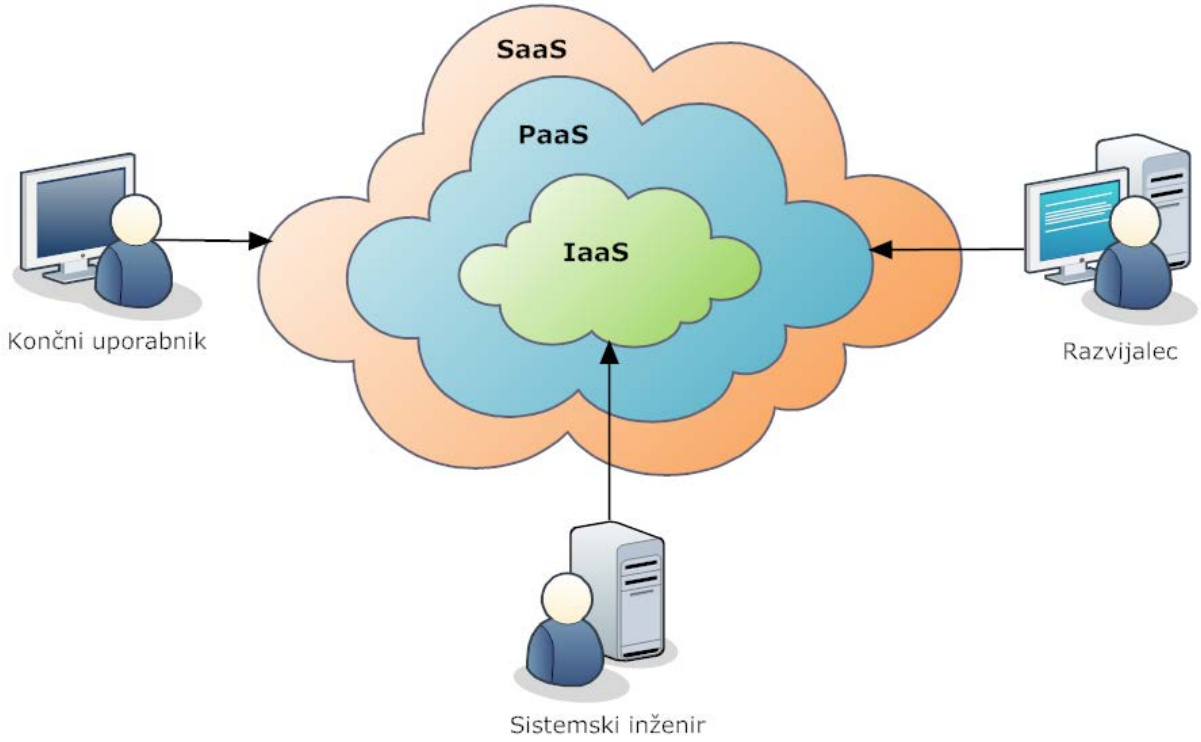
1.5 Modeli storitev

NIST definira tri modele storitev (*Service delivery models*) [13]:

- **Programska oprema kot storitev** (*Software as a Service, SaaS*): uporabnik ima možnost uporabljati ponudnikove aplikacije, ki tečejo na infrastrukturi oblaka. Aplikacije so dostopne na različnih odjemalcih preko uporabniškega vmesnika odjemalca kot je npr. spletni brskalnik (primer take aplikacije je Googlova spletna pošta Gmail). Uporabnik ne upravlja ali nadzira infrastrukture oblaka, kamor spadajo omrežje, strežniki, operacijski sistemi, shramba, niti ne posameznih aplikacij, z izjemo določenih, uporabniško specifičnih nastavitev.

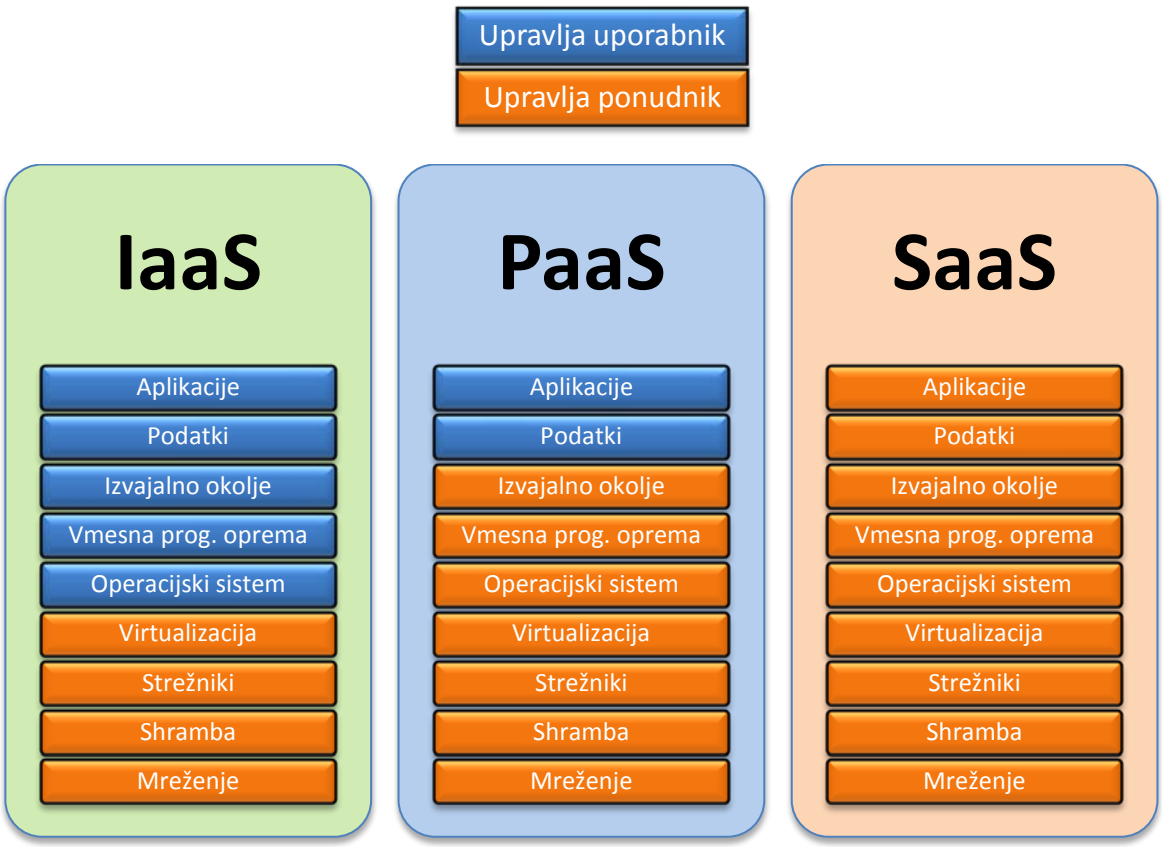
- **Platforma kot storitev** (*Platform as a Service, PaaS*): uporabnik ima možnost, da na infrastrukturi oblaka postavi lastno ali tujo aplikacijo, ki je bila razvita v programskih jezikih in orodjih, podprtih s strani ponudnika. Uporabnik ne upravlja ali nadzira infrastrukture oblaka, kamor spadajo omrežje, strežniki, operacijski sistemi, shramba, ima pa nadzor nad postavljenimi aplikacijami in nastavitvami okolja, v katerem aplikacija gostuje.

- **Infrastruktura kot storitev** (*Infrastructure as a Service, IaaS*): uporabniku se zagotavlja procesiranje, shramba, omrežje in drugi temeljni računalniški viri, ki omogočajo postavitev in izvajanje poljubne programske opreme, kar lahko vključuje operacijske sisteme in aplikacije. Uporabnik ne upravlja ali nadzira infrastrukture oblaka, ima pa nadzor nad operacijskimi sistemi, shrambo, postavljenimi aplikacijami in omejen nadzor nad določenimi mrežnimi komponentami (npr. požarni zid).



Slika 1.3. Modeli storitev računalništva v oblaku

Kot prikazuje slika 1.3, je vsak model storitev namenjen določenemu tipu uporabnika. Za IaaS je to sistemski inženir, za PaaS razvijalec, SaaS pa je storitev, ki je na voljo končnemu uporabniku. Vsi trije modeli storitev pa se razlikujejo tudi glede na razmerje med računalniškimi viri, s katerimi lahko upravljajo uporabniki storitev, ter viri, ki jih lahko upravlja le ponudnik storitev v oblaku. Primerjavo med tremi modeli storitev s tega vidika podaja spodnja slika 1.4 [14]:



Slika 1.4. Primerjava med IaaS, PaaS in SaaS

1.6 Modeli vzpostavitve oblaka

NIST podaja naslednje modele vzpostavitve oblaka (*Cloud deployment models*) [13]:

- **Zasebni oblak** (*Private cloud*): infrastruktura oblaka je na voljo izključno za neko organizacijo. Lahko jo upravlja organizacija sama ali tretja oseba in lahko obstaja na lokaciji (*on premise)* ali izven lokacije (*off premise*).
- **Skupni oblak** (*Community cloud*): infrastruktura oblaka je skupna večim organizacijam in podpira specifično skupnost, ki ima skupne interese (npr. poslanstvo, varnostne predpise, politiko in zahteve glede skladnosti). Upravljajo jo lahko organizacije, ki si jo delijo ali pa tretja oseba. Lahko obstaja na lokaciji ali izven lokacije.
- **Javni oblak** (*Public cloud*): infrastruktura oblaka je na voljo širši javnosti ali večji industrijski skupini in je v lasti organizacije, ki prodaja storitve v oblaku.

- **Hibridni oblak** (*Hybrid cloud*): infrastruktura oblaka je sestavljena iz dveh ali več oblakov (zasebni, skupni, javni), ki so vsak zase edinstvena entiteta, vendar jih povezuje standardizirana ali lastniška tehnologija, ki omogoča prenosljivost podatkov in aplikacij.

1.7 Prednosti

Na najvišjem nivoju lahko opredelimo naslednje prednosti storitev v oblaku, predvsem s poslovnega vidika [21]:

- **Agilnost** (*Agility*): storitve v oblaku omogočajo velike prihranke na času, saj omogočajo hitro dostavo zahtevanih storitev in s tem hitrejši ter učinkovitejši odziv organizacij na situacije kot so pritiski tekmecev, tržni izzivi, omejitve sredstev itd.

- **Osredotočenost na poslovanje** (*Business focus*): z uporabo najboljše storitve v oblaku, ki organizaciji omogoča pridobivanje informacij, idej, in povratnih informacij iz veliko večje množice virov (npr. stranke, partnerji) kot je bilo to možno prej, se lahko organizacija bolj osredotoči na samo poslovanje.

- **Nadzor stroškov in razpoložljivih sredstev** (*Cost and budget control*): storitve v oblaku omogočajo večji nadzor nad stroški in boljše upravljanje s sredstvi. Storitve se lahko zaračunavajo na podlagi naročnine ali pa na podlagi količine uporabljenih virov. V veliko primerih organizacijam tudi ni več treba zaposlovati ljudi, ki bi skrbeli npr. za sistemske posodobitve in varnostne kopije, zato lahko prihranijo na stroških zaposlovanja in upravljanja shrambe podatkov.

- **Skalabilnost in upravljanje kapacitet** (*Scalability and capacity management*): skalabilnost je sposobnost sistema, da se prilagaja številu zahtev in tako zagotavlja normalno delovanje tudi v primeru povečane obremenitve. Računalništvo v oblaku omogoča možnost skoraj takojšnjega povečanja zmogljivosti, prav tako pa tudi zmanjšanja, kadar se storitev ne uporablja, kar posledično pomeni tudi manjše stroške.

- **Optimizirana infrastruktura** (*Optimized infrastructure*): računalništvo v oblaku omogoča, da so zmogljivosti obstoječe infrastrukture na voljo večim uporabnikom. Gre za t.i. večnajemniški model (*Multitenancy model*), o katerem bo več govora v poglavju 2. Ta model je lahko implementiran na vsaki ali vseh plasteh arhitekture:

    - Virtualna plast: virtualizacija omogoča kreiranje specifičnih okolij za vsak proces, aplikacijo ali operacijski sistem. Večnajemniški model na tej plasti izolira vse kar je nad virtualno plastjo, omogoča pa deljeno uporabo skupnih virov, ki ležijo pod njo (mreža, procesor, pomnilnik, vhod/izhod in shramba podatkov).

    - Aplikacijska plast: posamezen uporabnik ima lahko uporabniški vmesnik aplikacije prilagojen po svoji meri.

  - Podatkovna plast: podatki za različne aplikacije, ki jih uporabljajo različni uporabniki so lahko shranjeni v eni sami podatkovni bazi, namesto da ima vsaka aplikacija svojo.

- **Mobilnost** (*Mobility*): z računalništvom v oblaku uporabniki lahko do storitev dostopajo kjerkoli in s pomočjo katerekoli naprave.

- **Vzdrževanje** (*Maintenance*): vzdrževanje aplikacij, ki so v oblaku, je lažje, ker ni potrebno, da so nameščene na računalniku vsakega uporabnika. Lažje je tudi zagotavljati njihovo podporo in posodobitve, ker je uveljavljanje sprememb na strani odjemalcev skoraj takojšnje.

Poglavje 2

Platforma kot storitev

2.1 Kaj je platforma

V splošnem si nek abstrakten model platforme za razvoj aplikacij lahko predstavljamo kot skupek naslednjih treh delov [2]:

- **Temeljni del:** skoraj vsaka aplikacija potrebuje na računalniku, kjer se izvaja, neko platformsko programsko opremo. Običajno gre za operacijski sistem ter podporne storitve kot so standardne knjižnice in shramba.
- **Infrastrukturne storitve:** v sodobnem porazdeljenem okolju aplikacije pogosto uporabljajo storitve, ki se nahajajo na drugih računalnikih. Gre npr. za dostop do oddaljene shrambe podatkov, povezovalne storitve, storitve identifikacije itd.
- **Aplikacijske storitve:** s tem ko aplikacije postajajo vedno bolj storitveno usmerjene, so njihove funkcije na razpolago tudi drugim aplikacijam. Čeprav te aplikacije obstajajo z namenom, da omogočajo storitve končnim uporabnikom, so prav tako del platforme za razvoj aplikacij.

V povezavi z razvojno platformo je potrebno omeniti tudi razvojna orodja. Ta razvijalcem omogočajo razvoj aplikacij z uporabo vseh treh zgoraj omenjenih delov platforme.

Za boljšo predstavo podajmo konkretne primere za posamezen sestavni del razvojne platforme:

- **Temeljni del:**
    - Operacijski sistem: npr. Windows ali Linux

  - Lokalne podporne storitve: npr. ogrodje .NET in Java EE aplikacijski strežniki, podatkovne baze (MySQL, Oracle DBMS, IBM DB2) za shrambo podatkov

- **Infrastrukturne storitve:**
  - Shramba: npr. dostop do oddaljenih podatkovnih baz
  - Povezovalne storitve: gre za povezovanje aplikacij, npr. IBM WebSphere Process Server, Microsoft BizTalk Server
  - Storitve identifikacije: Microsoft Active Directory, LDAP in drugi mehanizmi za avtentikacijo
- **Aplikacijske storitve:** različne organizacije uporabljajo različne aplikacije, ki jih lahko razdelimo v dve širši kategoriji:
  - Programski nabori: gre za poslovno programsko opremo kot sta SAP in Microsoft Dynamics
  - Aplikacije po meri

2.2 Tradicionalni model

Razvoj in izvajanje aplikacij je bilo pred pojavom računalništva v oblaku kompleksno in drago opravilo [23]. Vsaka aplikacija je potrebovala ustrezno strojno opremo, operacijski sistem, podatkovno bazo, vmesno programsko opremo (*Middleware*), spletne strežnike in ostalo potrebno programsko opremo. Za sam razvoj aplikacije so morali razvijalci uporabljati kompleksna razvojna okolja, kot npr. J2EE in .NET. Poleg tega, je bila potrebna še ekipa strokovnjakov s področja mrež, podatkovnih baz in sistemske administracije. V primeru nove poslovne zahteve, ki je običajno zahtevala tudi spremembo aplikacije, je bil potreben dolg razvojni, testni in postavitveni cikel.

Velika podjetja so ponavadi potrebovala posebne prostore za svoje podatkovne centre in njihove nadomestne lokacije za obnavljanje podatkov v primeru nesreč. Potrebne so bile tudi ogromne količine električne energije za napajanje strežnikov in sistemi za njihovo hlajenje.

2.3 Novi model

Računalništvo v oblaku prinaša tudi platforme za razvoj in izvajanje aplikacij. Gre za model platforme kot storitve ali PaaS, katerega »uradno« definicijo po NISTU smo podali že v prejšnjem poglavju *Računalništvo v oblaku*.

Model PaaS omogoča razvijalcem programske opreme in IT oddelkom, da se namesto ukvarjanja s kompleksno infrastrukturo raje osredotočijo na sam razvoj aplikacije. Posledično to pomeni tudi večjo stroškovno učinkovitost, saj lahko sredstva, ki so bila prej namenjena za

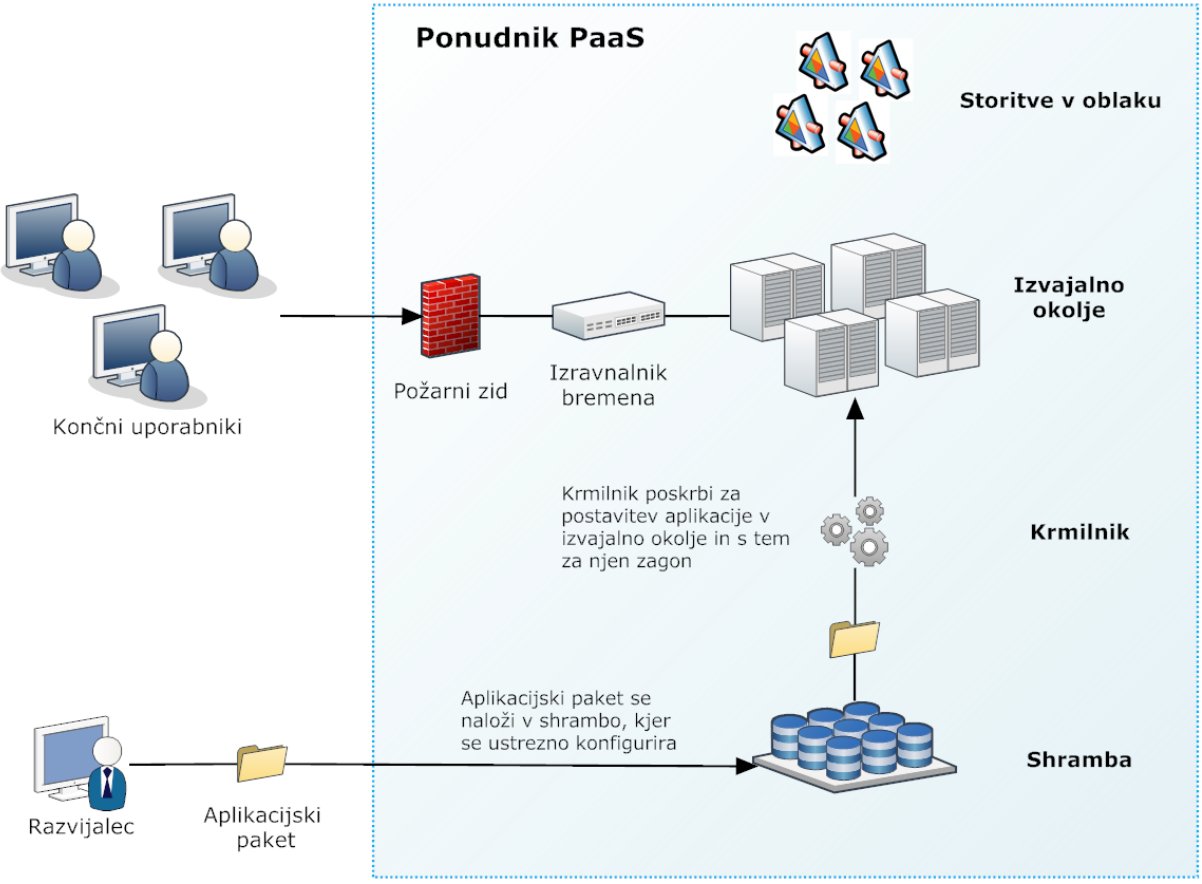
vzdrževanje in podporo infrastrukture, preusmerijo v razvoj aplikacij. Te so pravzaprav tiste, ki dajejo koristen prispevek poslovanju.

Tako kot uporabljajo in plačujejo storitve elektrike in vode, tudi v tem primeru razvijalci storitve platforme koristijo in plačujejo toliko, kolikor jih potrebujejo, brez da bi imeli opravka z vsemi kompleksnimi zadevami, ki se odvijajo v ozadju [23].

2.4 Model platforme kot storitve

Model PaaS je pristop, s katerim ponudnik ponuja funkcionalnosti platforme za razvoj aplikacij kot storitev v oblaku. Uporabnik modela je razvijalec aplikacije oziroma lastnik programske opreme, ki želi, da bi bila njegova aplikacija dostopna končnim uporabnikom prek spleta. Ponudnik uporabniku v zameno za plačilo nudi razvojno in izvajalno okolje za njegove aplikacije ter ostale storitve v oblaku.

Posplošen model PaaS prikazuje slika 2.1 [20]:



Slika 2.1. Model PaaS

Razvojno okolje se lahko nahaja v oblaku, lahko pa razvijalec aplikacijo razvija na svojem lokalnem računalniku in jo potem v obliki aplikacijskega paketa naloži na platformo PaaS.

Aplikacijski paket vsebuje samo tisto, kar je razvil lastnik programske opreme s pomočjo razvojnega okolja, ki ga podpira PaaS ponudnik. Pri tem lahko koristi storitve v oblaku, ki so mu v okviru izbrane rešitve PaaS na voljo. Ponavadi so to storitve, ki so namenjene shranjevanju podatkov, povezovanju z drugimi aplikacijami ter implementaciji varnostnih mehanizmov kot sta avtentikacija in avtorizacija.

Aplikacijski paket, ki vsebuje kodo aplikacije, se najprej naloži v shrambo. Na podlagi konfiguracijske datoteke, ki je vključena v aplikacijskem paketu, se aplikacija ustrezno konfigurira (npr. razvijalec določi število instanc aplikacije). Krmilnik nato na podlagi konfiguracijske datoteke kreira in zažene določeno število instanc aplikacije in tako aplikacijo preda v izvajalno okolje platforme PaaS. Za enakomerno obremenitev aplikacije skrbi izravnalnik bremena (*Load Balancer*), ki zahteve končnih uporabnikov porazdeli po več instancah aplikacije.

2.4.1 Večnajemniški model

Za razliko od tradicionalnih modelov računanja, kjer so bili računalniški viri dodeljeni vsakemu posameznemu uporabniku, računalništvo v oblaku in s tem tudi model PaaS temeljita na modelu, imenovanem večnajemniški model (*Multitenancy model*) [22]. Glavni koncept modela je deljenje skupnih virov, kar pomeni, da več uporabnikov ali t.i. najemnikov (*Tenants*) lahko hkrati uporablja iste vire.

Večnajemniški model je izboljšan naslednik modela ASP (*Application Service Provider*) iz 90-ih let [22]. Model ASP posameznemu najemniku preko omrežja omogoča dostop do aplikacij na lokaciji - podatkovnem centru. Najemniki si delijo samo infrastrukturo podatkovnega centra, kamor spada napajanje, hlajenje in sama lokacija, vsak zase pa uporabljajo svojo instanco aplikacije, vmesnega sloja programske opreme, operacijskega sistema in svoj strežnik. Ta pristop je najbolj primeren v primerih, ko je potrebna zelo velika stopnja prilagojevanja najemnikom in izoliranosti med njimi. Po drugi strani pa je njegova slabost velika stroškovna neučinkovitost postavitve in vzdrževanja strojne ter programske opreme za vsakega najemnika posebej.

2.4.1.1 Implementacija večnajemniškega modela

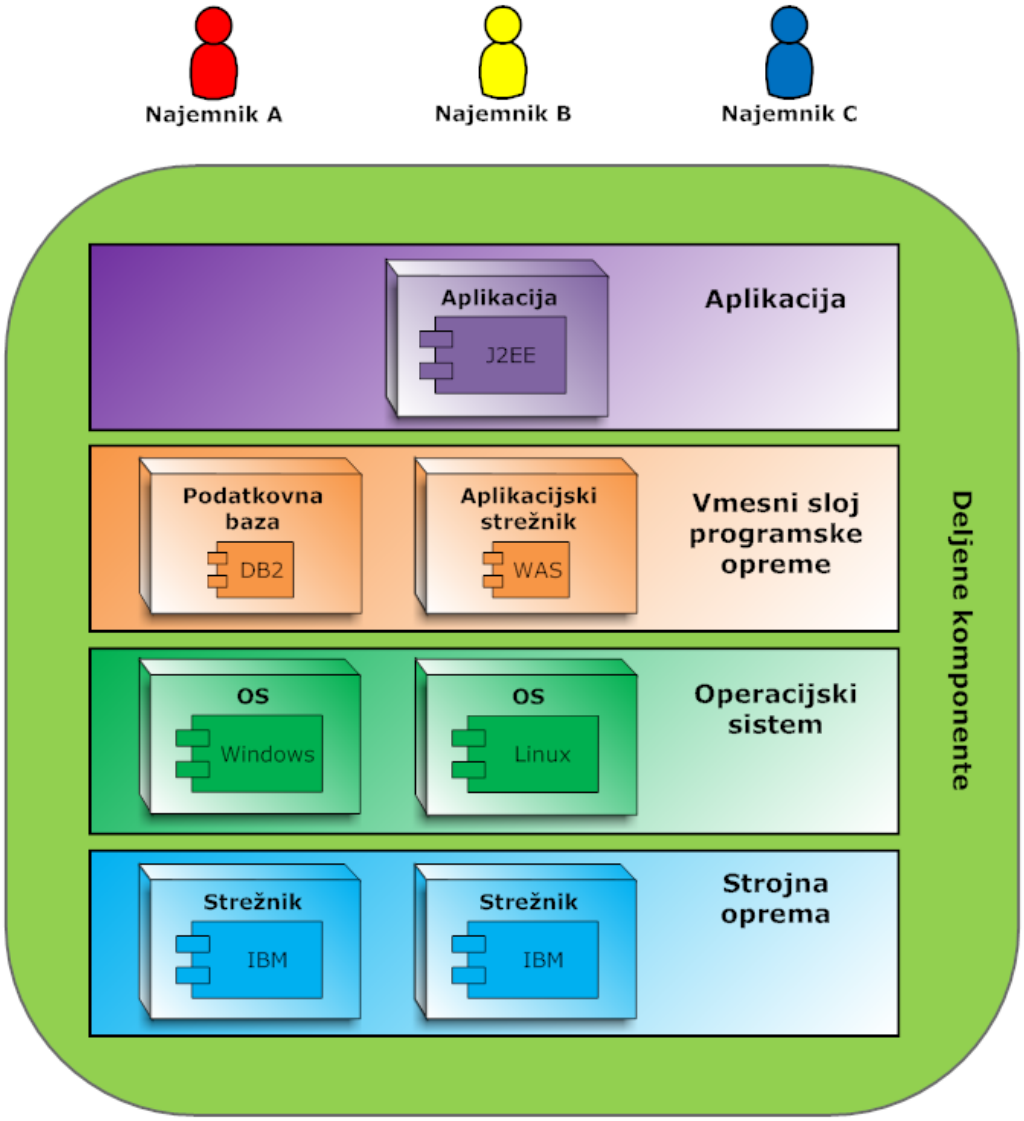
Obstaja več pristopov k implementaciji večnajemniškega modela, delimo pa jih v dve skupini [7]:

- **Deljenje vmesnega sloja programske opreme** (pod vmesni sloj programske opreme spadajo npr. aplikacijski strežniki in podatkovne baze):
    - **Pristop 1:** Ena sama instanca aplikacije
    - **Pristop 2:** Več instanc aplikacije v deljenem naslovnem prostoru
    - **Pristop 3:** Več instanc aplikacije v ločenem naslovnem prostoru

- **Virtualizacija**
    - **Pristop 4:** Virtualizacija

Pristop 1: Ena sama instanca aplikacije
Pristop prikazuje slika 2.2 [7]. Najemniki si delijo operacijski sistem, strežnike, vmesni sloj programske opreme in eno samo instanco aplikacije. To omogoča, da si npr. prilagodijo uporabniški vmesnik aplikacije (izgled in podatke).

V tem primeru mora razvijalec poskrbeti, da njegova aplikacija izolira podatke in nastavitve posameznega najemnika od drugih najemnikov. Eden izmed načinov je, da se instanci aplikacije kot parameter poda unikaten ID najemnika. Ta ID se pripne vsakemu objektu in operacijam, ki jih aplikacija nudi. Če aplikacija uporablja tabele podatkovne baze, se vsaki tabeli doda še stolpec, ki vsebuje ID.

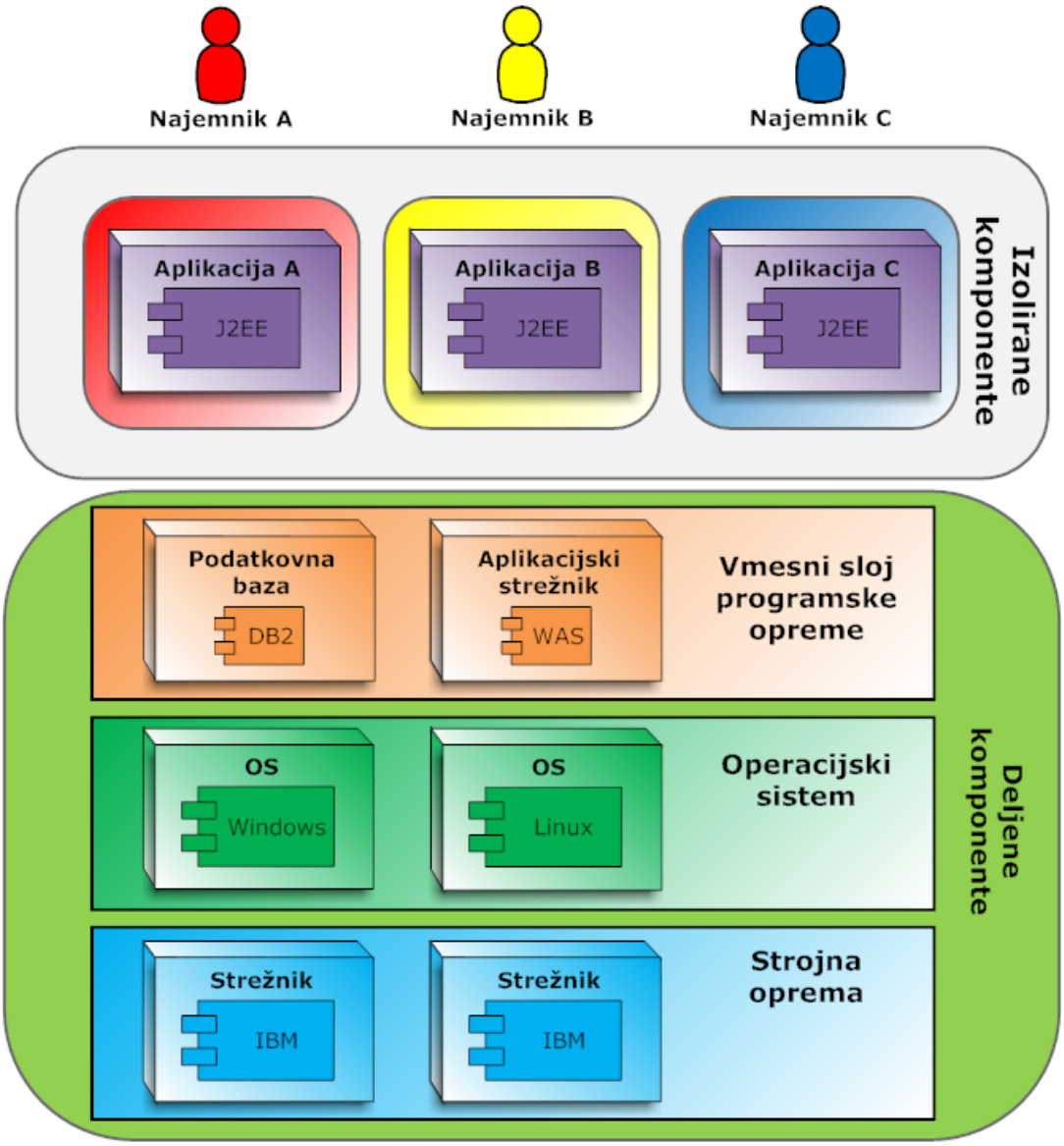


Slika 2.2. Ena sama instanca aplikacije

Pristop 2: Več instanc aplikacije v deljenem naslovnem prostoru

Pristop prikazuje slika 2.3 [7]. Najemniki si delijo operacijski sistem, strežnike, vmesni sloj programske opreme, vsak pa uporablja svojo instanco aplikacije. Ker se vmesni sloj programske opreme deli med najemnike, vsi uporabljajo isti proces operacijskega sistema oziroma isti naslovni prostor. Ko se nov najemnik prijavi, se zanj kreira nova kopija aplikacije, ki se ji pripne najemnikov ID. Na podoben način se mu v podatkovni bazi dodeli lastna kopija podatkovne tabele.

Ta model zahteva, da je izolacija najemnikov implementirana na nivoju vmesnega sloja programske opreme.

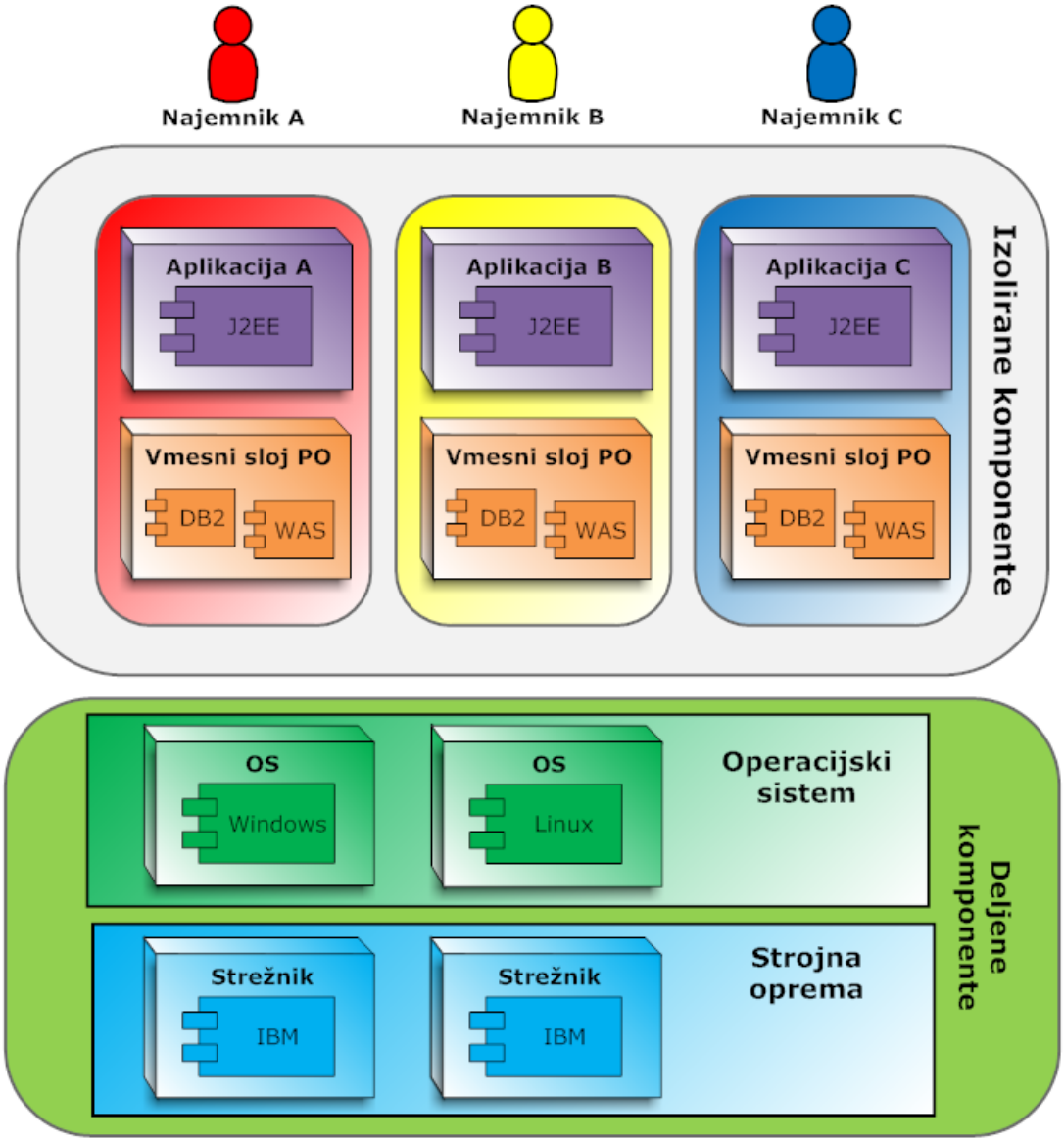


Slika 2.3. Več instanc aplikacije v deljenem naslovnem prostoru

Pristop 3: Več instanc aplikacije v ločenem naslovnem prostoru

Pristop prikazuje slika 2.4 [7]. Najemniki si delijo operacijski sistem in strežnike, vsak pa uporablja svojo instanco vmesnega sloja programske opreme in svojo instanco aplikacije. Ker vsak najemnik uporablja svojo instanco vmesnega sloja programske opreme, mu pripada lastna množica procesov operacijskega sistema oziroma lastni naslovni prostor.

Ta način zahteva, da izolacijo med najemniki vzdržuje operacijski sistem in podpira manj gostujočih najemnikov na istem strežniku kot pristopa 1. in 2. Po drugi strani pa omogoča večjo stopnjo izolacije med najemniki. Problem pri tem modelu je, da se lahko zgodi, da končni uporabniki nekega najemnika zasedejo celotno procesorsko moč in pomnilnik fizičnega strežnika.

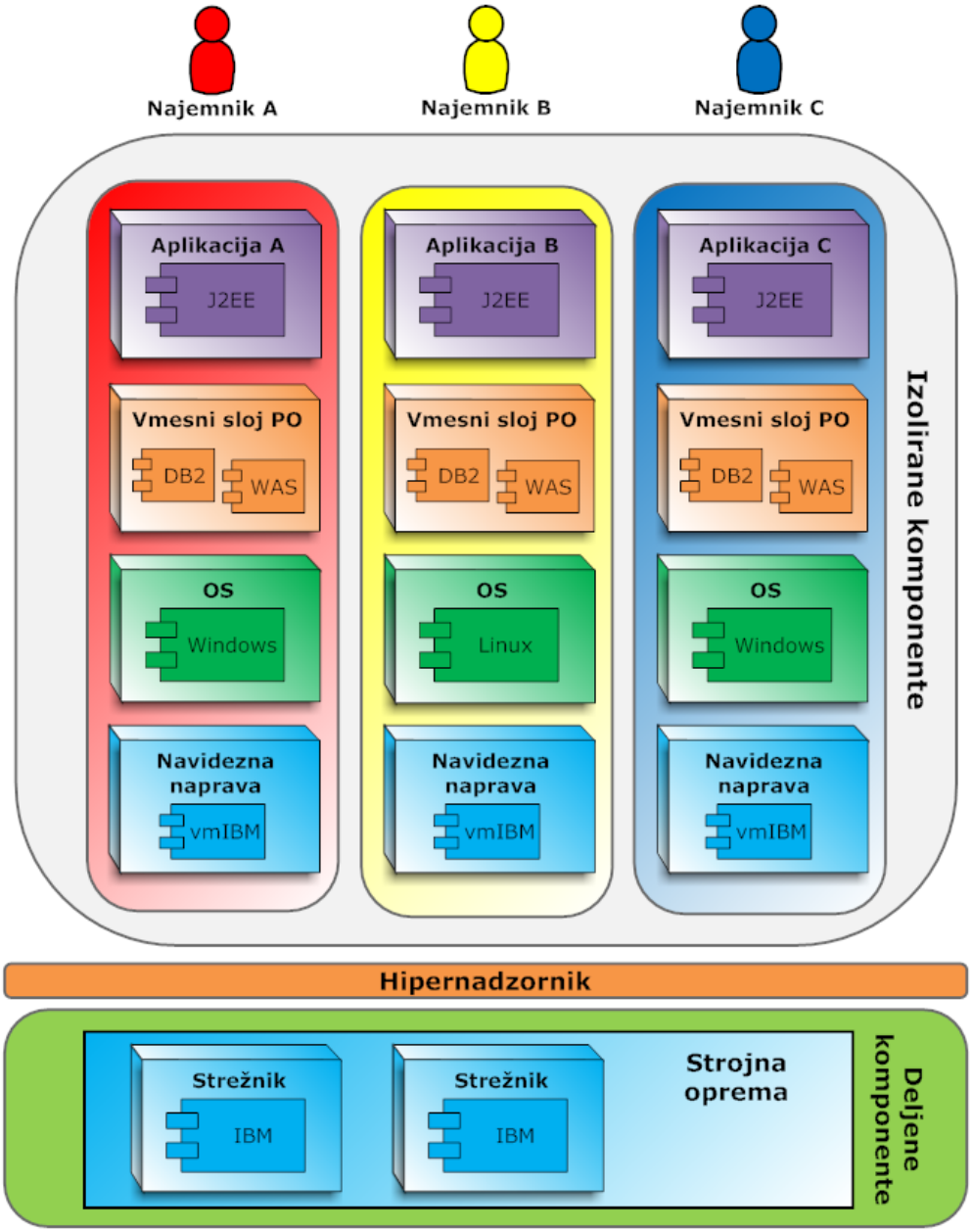


Slika 2.4. Več instanc aplikacije v ločenem naslovnem prostoru

Pristop 4: Virtualizacija

Pristop je prikazan na sliki 2.5 [7]. Najemniki si delijo fizične strežnike, vsak pa uporablja svojo navidezno napravo, na kateri tečejo ločena aplikacija, ločen vmesni sloj programske opreme in ločen operacijski sistem. Za preslikavo med fizičnim strežnikom in navidezno napravo poskrbi nizkonivojska plast programske opreme imenovana hipernadzornik (*Hypervisor*).

Za razliko od prejšnjih pristopov virtualizacija zahteva skorajda nič pisanja kode za implementacijo večnajemniškega modela. Ponudnik storitve namreč vsakemu najemniku dodeli svojo navidezno napravo.



Slika 2.5. Virtualizacija

2.4.1.2 Primerjava med pristopi z vidika ponudnika in razvijalca

ASP in ostali pristopi

Če primerjamo opisane štiri pristope z modelom ASP, so z vidika nakupovanja, postavitve ter vzdrževanja strojne in programske opreme le-ti stroškovno bolj učinkoviti, ker temeljijo na deljenju virov.

Po drugi strani pa je model ASP najcenejši z vidika razvijalca, ker za implementacijo večnajemniškega modela ni potreben poseg v aplikacijo. Prav tako model ASP omogoča najvišjo možno izolacijo in prilagodljivost med najemniki.

Štirje pristopi

Med samimi štirimi pristopi so z vidika ponudnika storitev najcenejši prvi trije, ker deljena uporaba strojne in programske opreme omogoča nižje stroške nabave, postavitve in vzdrževanja. Prav tako omogočajo največje število najemnikov, ki lahko gostujejo na enem fizičnem strežniku, s tem pa posledično največjo skalabilnost.

Z vidika razvijalca predstavlja najdražjo možnost prvi pristop, ker je za implementacijo večnajemniškega modela potreben največji poseg v aplikacijo, prav tako pa največja količina znanja in izkušenj razvijalca. Na nasprotni strani je za razvijalca najcenejši četrti pristop.

Zgoščeno primerjavo med vsemi opisanimi pristopi podaja spodnja preglednica 2.1:

Preglednica 2.1. Primerjava med štirimi pristopi implementacije večnajemniškega modela

	Deljenje vmesnega sloja programske opreme	Virtualizacija
Prednosti	• *hitra skalabilnost (večje število možnih najemnikov na posamezen fizični strežnik)* • *stroškovna učinkovitost z vidika ponudnika PaaS*	• *ni potreben poseg v aplikacijo* • *večja stopnja izolacije in prilagajanja* • *lažja prenosljivost aplikacij na drugega ponudnika PaaS* • *bolj enostavno varnostno shranjevanje in obnavljanje prilagojenih podatkov za vsakega posameznega najemnika v primeru nesreč* • *manjši stroški z vidika razvijalca*
Slabosti	• *večji stroški z vidika razvijalca* • *pristop 1 zahteva velik poseg v aplikacijo in izkušene razvijalce* • *posledica zgornje točke je tudi daljši čas za postavitev aplikacije v produkcijo* • *večja kompleksnost pri zagotavljanju varnostnega shranjevanja in obnavljanja prilagojenih podatkov najemnikov v primeru nesreč*	• *manjša skalabilnost* • *večji stroški z vidika ponudnika PaaS*

2.4.2 Glavne karakteristike PaaS

Celovita rešitev PaaS naj bi vsebovala vsaj naslednje glavne funkcionalnosti [9,19]:

- **Načrtovanje in razvoj aplikacij**
- **Testiranje in postavitev aplikacij**
- **Povezovanje aplikacij**
- **Spremljanje**
- **Skalabilnost**
- **Zanesljivost**
- **Varnost**
- **Večnajemniški model**
- **Shramba**

- **Ostale storitve**

Načrtovanje in razvoj aplikacij
Gre za možnost načrtovanja, razvoja in testiranja aplikacij ter uporabniških vmesnikov s pomočjo razvojnih orodij. Tu imamo lahko na voljo dve možnosti:

- razvojno okolje je lahko del platforme v oblaku in do njega lahko dostopamo preko spletnega brskalnika,
- razvijalec aplikacijo razvija na lokalnem razvojnem okolju in jo nato v obliki aplikacijskega paketa naloži na platformo v oblaku (to lahko naredi kar znotraj razvojnega okolja).

Nekateri ponudniki PaaS omogočajo migracijo obstoječe aplikacijske kode v oblak, ostali pa razvijajo nove programske jezike in nova okolja, ki so specifična za oblak.

Testiranje in postavitev aplikacije
Gre za možnost testiranja, zapakiranja in postavitve aplikacij.

Izvajalno okolje
Platforma PaaS nudi izvajalno okolje za aplikacije.

Povezovanje
Gre za možnost povezovanja med aplikacijami ter med aplikacijami in zunanjimi spletnimi rešitvami ter podatkovnimi bazami. Aplikacije naj ne bi delovale izolirano od ostalih aplikacij, čeprav se izvajajo v oblaku. Povezovalne storitve omogočajo, da aplikacije lahko komunicirajo med sabo.

Spremljanje
Gre za spremljanje aktivnosti aplikacij in uporabnikov, da lahko razvijalci bolje razumejo delovanje svojih aplikacij in izdelajo popravke.

Skalabilnost
Skalabilnost mora biti v rešitvah PaaS vgrajena brez dodatnega razvoja, kompleksnih nastavitev ali drugih stroškov. Razvijalec mora imeti možnost, da v primeru povečanega števila zahtev enostavno in skorajda takoj poveča zmogljivost svoje aplikacije. Na primer, če potrebuje tri instance spletnega uporabniškega vmesnika za potrebe obvladovanja pričakovane obremenitve, potem lahko to definira v konfiguracijski datoteki, izvajalno okolje platforme pa bo na podlagi te datoteke samodejno ustvarilo tri instance spletnega uporabniškega vmesnika.

Zanesljivost
Ponudniki rešitev PaaS morajo v čim večji meri zagotoviti zanesljivo delovanje platform tudi v primeru napak in naravnih katastrof.

Varnost
Eden izmed najpomembnejših dejavnikov pri rešitvah PaaS je tudi sposobnost zagotavljanja različnih vidikov varnosti. Tu gre predvsem za zagotavljanje razpoložljivosti platforme in

posledično aplikacij ter podatkov, zagotavljanje integritete podatkov, zaupnosti in mehanizmov za avtentikacijo ter avtorizacijo uporabnikov [10]. Na tem mestu velja omeniti tudi pomembnost prisotnosti dogovora o nivoju storitev (*Service Level Agreement, SLA*). Gre za dogovor med ponudnikom storitve in uporabnikom, ki med drugim opredeljuje nivo zagotavljanja varnosti in razpoložljivosti storitev ter ravnanje v primeru napak oziroma nesreč.

Večnajemniški model
Večnajemniški model je osnovni temelj računalništva v oblaku in s tem tudi modela PaaS.

Shramba
Gre za možnost shranjevanja podatkov aplikacije v obliki podatkovne baze ali datotečnega sistema.

Poslovne aplikacije po meri imajo ponavadi različne potrebe po shrambi. Platforma v oblaku mora biti sposobna nuditi datotečne storitve za osnovno shranjevanje, sporočilne vrste in napredno shranjevanje. Osnovno shranjevanje je potrebno za velike binarne objekte (*Binary Large Objects, BLOBs*) in notranje podatkovne strukture. Sporočilne vrste se lahko uporablja za asinhrono komunikacijo z ostalimi komponentami. Napredno shranjevanje pa omogoča vse zmogljivosti napredne podatkovne baze, ki so potrebne za skalabilno dostopanje do podatkov in izdelavo poročil.

Ostale storitve
PaaS nudi tudi skupek ostalih storitev, ki rešujejo nek višjenivojski problem. Ponavadi gre za storitve, ki so tipične za nekega ponudnika PaaS (npr. Google Gmail, ki omogoča storitve e-pošte).

2.4.3 Tipi platform PaaS

Glede na različen obseg funkcionalnosti, ki jih ponujajo, imamo na voljo sledeče tipe platform PaaS [26]:

- **Platforme za nadgrajevanje obstoječih aplikacij:** gre za platforme, ki omogočajo nadgrajevanje oziroma spreminjanje obstoječih aplikacij SaaS.
- **Samostojne platforme:** gre za platforme, ki nudijo neko splošno razvojno okolje in niso vezane na specifične aplikacije SaaS.
- **Platforme za postavitev aplikacij:** gre za platforme, ki nudijo samo storitve, ki omogočajo gostovanje aplikacij, npr. storitve zagotavljanja varnosti in skalabilnosti na zahtevo, ne ponujajo pa okolja za razvoj, razhroščevanje in testiranje aplikacij.
- **Odprte platforme:** gre za platforme, ki razvijalcu omogočajo prosto izbiro programskega jezika, podatkovne baze, operacijskega sistema itd.

2.4.4 Razlike med tradicionalno razvojno platformo in platformo PaaS

Funkcionalnosti platforme PaaS se razlikujejo od funkcionalnosti tradicionalnih razvojnih platform. Razlike so sledeče [12]:

- **Razvojna orodja, ki podpirajo večnajemniški model:** tradicionalna razvojna orodja so namenjena enemu uporabniku, medtem ko naj bi razvojna orodja platform PaaS podpirala več uporabnikov, vsak od njih pa ima lahko več aktivnih projektov.
- **Arhitektura, ki podpira večnajemniški model:** v začetni fazi tradicionalnega razvoja aplikacij se ponavadi ne ukvarjamo s skalabilnostjo, ta je prepuščena sistemskim administratorjem v fazi postavitve. Pri PaaS pa mora biti skalabilnost na nivoju aplikacije in podatkov že vgrajena.
- **Vgrajeno upravljanje:** tradicionalne razvojne rešitve ponavadi nimajo možnosti spremljanja aplikacije med izvajanjem, medtem ko mora biti pri PaaS ta funkcija že vgrajena v platformo.
- **Vgrajeno zaračunavanje:** platforme PaaS omogočajo zaračunavanje storitev na podlagi uporabe.

S tem, ko se platforme selijo v oblak, imamo opravka tudi z določenimi omejitvami glede njihove prilagodljivosti. Spodnja preglednica 2.2 prikazuje primerjavo med prilagodljivostjo tradicionalnih razvojnih platform in platform PaaS:

Preglednica 2.2. Primerjava med tradicionalno platformo in platformo PaaS z vidika prilagodljivosti

Vrsta področja	Tradicionalne platforme	Platforme PaaS
Programski jeziki	*Podpirajo več različnih programskih jezikov*	*Izbira omejena z izbranim ponudnikom rešitve PaaS*
Aplikacijski strežniki	*Podpirajo več različnih ponudnikov*	*Izbira omejena z izbranim ponudnikom rešitve PaaS*
Podatkovne baze	*Podpirajo več različnih ponudnikov*	*Izbira omejena z izbranim ponudnikom rešitve PaaS*
Strežniki in navidezne naprave	*Podpirajo več različnih ponudnikov*	*Izbira omejena z izbranim ponudnikom rešitve PaaS*
Shramba	*Podpirajo več različnih ponudnikov*	*Izbira omejena z izbranim ponudnikom rešitve PaaS*

2.4.5 Prednosti

Poleg vseh prednosti, ki jih prinaša računalništvo v oblaku, lahko definiramo še sledeče prednosti uporabe modela PaaS:

- **Celovito razvojno okolje:** PaaS razvijalcem aplikacij omogoča dostop do celovitega razvojnega okolja, ki omogoča izdelavo celovitih aplikacij vse od podatkov do uporabniškega vmesnika.
- **Podpora celotnemu razvojnemu ciklu aplikacije:** PaaS podpira celoten cikel razvoja aplikacije, od načrtovanja, implementacije in testiranja, do postavitve in vzdrževanja.
- **Hitra distribucija programske opreme:** aplikacija, ki jo razvijalec naloži na oblak, je takoj dostopna končnim uporabnikom po vsem svetu.
- **Manjša kompleksnost in nizki stroški razvoja in postavitve aplikacij:** s pomočjo PaaS imajo tudi majhni razvijalci in start-up podjetja možnost, da postavijo spletne aplikacije brez velikih stroškov in kompleksnosti, povezanih z nakupovanjem in nameščanjem strežnikov.
- **Nižji stroški izvajanja aplikacij:** razvijalci oziroma lastniki aplikacij plačajo porabo le tistih virov, ki jih aplikacija tekom izvajanja dejansko zaseda.
- **Skalabilnost:** Takojšnja skalabilnost aplikacij v primeru povečanja zahtev.
- **Osredotočenost na razvoj:** razvijalci so osredotočeni na sam razvoj aplikacije in ne na vzdrževanje infrastrukture, ki je potrebna za izvajanje aplikacije.
- **Možnost hitre in enostavne posodobitve aplikacije:** s tem ko uporabniki dejansko uporabljajo samo eno fizično instanco aplikacije v oblaku, ni potrebno, da se posodobitev izvede za vsakega uporabnika posebej.
- **Povečana zanesljivost in varnost podatkov:** za podjetja, ki nimajo ustreznih načrtov za varnostno kopiranje podatkov in okrevanje v primeru odpovedi strojne opreme in naravnih nesreč, je PaaS dobrodošla rešitev.

2.4.6 Slabosti

Kljub temu, da prinaša PaaS veliko prednosti, se pri uporabi tega modela srečujemo z naslednjimi problemi in ovirami:

- **Omejena fleksibilnost:** razvoj je omejen na funkcionalnosti, ki jih ponuja ponudnikova platforma.
- **Slaba povezljivost:** s tem, ko imamo z izbiro določenega ponudnika PaaS na voljo le določene programske jezike in drugo določeno tehnologijo za razvoj in postavitev

aplikacij, je oteženo tudi prenašanje aplikacije na drugo platformo. Pojavi se problem t.i. priklenitve na ponudnika (*Vendor lock-in*).

- **Zanesljivost in varnost:** kljub temu, da z uporabo PaaS določena podjetja lahko pridobijo na večji zanesljivosti in varnosti podatkov, pa je po drugi strani to zelo pereč problem. Podatki, predvsem tisti, ki pripadajo večjim organizacijam, so lahko zelo občutljive narave in s tem, ko se nahajajo v oblaku, so izpostavljeni zlorabam. Ponudnik PaaS mora na tem mestu poskrbeti za najvišji nivo zaščite podatkov in njihovo varnostno kopiranje ter obnavljanje v primeru odpovedi delovanja.

2.5 Aplikacijski programski vmesniki API

Pri večini platform, ki so na voljo v oblaku, gre za aplikacijska ogrodja (*Application Frameworks*) [5]. Ta ogrodja tipično omogočajo splošne storitve, kot so uporabniški vmesniki, shramba in podatkovne baze, do njih pa je možno dostopati le preko aplikacijskih programskih vmesnikov (*Application Programming Interface, API*).

API-ji, ki jih ponudniki oblakov dajo na voljo razvijalcem, so glavni mehanizem, ki se uporablja za razvoj rešitev v oblaku [25]. Večina ponudnikov podaja svoje API-je, ki so ponavadi dobro dokumentirani, vendar tudi edinstveni v svoji implementaciji, kar posledično onemogoča njihovo interoperabilnost. Nekateri ponudniki pa se poslužujejo API-jev tretjih oseb in tako je na voljo tudi veliko razvijajočih se odprtih standardov, kot je npr. OCCI (*Open Cloud Computing Interface*).

Aplikacijski programski vmesniki v oblaku delujejo na štirih različnih nivojih, umestimo pa jih lahko v pet osnovnih kategorij.

2.5.1 Nivoji API-jev

Obstajajo štirje različni nivoji API-jev [5]. Vsak nivo zahteva od razvijalca, da se osredotoči na različna opravila in podatkovne strukture.

- **NIVO 1 – Prenosni nivo** (*The wire*): če storitev temelji na REST arhitekturi, mora razvijalec ustvariti primerne HTTP glave, vsebino in odpreti HTTP povezavo na storitev. REST storitev vrne podatke skupaj s kodo HTTP odgovora. Zaradi enostavne narave večine REST storitev, je pisanje kode na tem nivoju relativno učinkovito.

  Če storitev temelji na SOAP protokolu, mora razvijalec ustvariti SOAP ovojnico, dodati primerne SOAP glave in v telo SOAP ovojnice vstaviti vsebino. SOAP storitev vrne odgovor kot SOAP ovojnico, ki vsebuje rezultat zahteve. Delo s SOAP storitvami zahteva razčlenjevanje XML vsebine v ovojnici. Iz tega razloga se na večino SOAP storitev sklicujemo preko višjenivojskih API-jev.

- **NIVO 2 – Jezikovno-specifični nabori orodij** (*Language-Specific Toolkits*): razvijalci na tem nivoju za delo s SOAP ali REST zahtevami uporabljajo nabor orodij,

ki je specifičen za nek jezik. Čeprav je pri razvoju še vedno potrebna pozornost na obliko in strukturo podatkov, ki se prenašajo, za večino tu poskrbi nabor orodij.

- **NIVO 3 – Storitveno-specifični nabori orodij** (*Service-Specific Toolkits*): razvijalec ima za delo z določeno storitvijo na voljo visokonivojski nabor orodij. Na tem nivoju se lahko razvijalec osredotoči na poslovne objekte in poslovne procese. Ker je njegova pozornost usmerjena na podatke in procese, ki so pomembni za organizacijo, je njegova produktivnost večja, kot če bi se ukvarjal s protokolom na prenosnem nivoju.

- **NIVO 4 – Storitveno-neodvisni nabori orodij** (*Service-Neutral Toolkits*): gre za najvišji API nivo. Razvijalec ima na voljo nek skupen vmesnik za več ponudnikov storitev v oblaku. Podobno kot na nivoju 3, se razvijalec osredotoča na poslovne objekte in poslovne procese, a s to razliko, da se mu ni treba ukvarjati s tem, na katero storitev v oblaku se nanašajo. Aplikacija, razvita s storitveno-neodvisnimi nabori orodij, se lahko prenese na oblak drugega ponudnika z minimalnimi spremembami na sami kodi.

2.5.2 Kategorije API-jev

Aplikacijski programski vmesniki se delijo na pet kategorij [5]:

- **KATEGORIJA 1 – Splošno programiranje:** sem spadajo običajni aplikacijski programski vmesniki v C#, PHP, Java, itd. V tej kategoriji ni nič, kar bi bilo specifično za oblak.

- **KATEGORIJA 2 – Postavitev:** gre za aplikacijske programske vmesnike, ki služijo postavitvi aplikacij v oblak. Poleg raznih paketnih tehnologij, ki so specifične za oblak, sem spadajo tradicionalni mehanizmi kot npr. .NET programske komponente in WAR datoteke.

- **KATEGORIJA 3 – Storitve v oblaku:** sem spadajo aplikacijski programski vmesniki, ki delujejo s storitvami v oblaku. Kot je bilo že omenjeno v prejšnjem podpoglavju, so API-ji za storitve v oblaku lahko storitveno-specifični ali storitveno-neodvisni. API-ji v tej kategoriji se delijo naprej v naslednje podkategorije, in sicer na API-je za:
    - storitve shranjevanja podatkov v oblaku,
    - podatkovne baze v oblaku,
    - sporočilne vrste v oblaku,
    - ostale storitve v oblaku.

- **KATEGORIJA 4 – Upravljanje slik navideznih naprav in infrastrukture:** tu se nahajajo aplikacijski programski vmesniki za podrobno upravljanje slik navideznih naprav in infrastrukture. API-ji za navidezne naprave podpirajo nalaganje, zagon,

zaustavitev, resetiranje in brisanje slik. API-ji za upravljanje infrastrukture pa služijo za nadzor nad požarnimi zidovi, upravljanjem vozlišč, upravljanjem omrežij in izravnavanjem obremenitve.

- **KATEGORIJA 5 – Notranji vmesniki:** sem spadajo aplikacijski programski vmesniki, ki služijo kot notranji vmesniki med različnimi deli infrastrukture oblaka. To so API-ji, ki bi jih uporabili v primeru, da bi želeli zamenjati ponudnika shrambe podatkov.

2.5.3 Vloge razvijalcev

Razvijalci imajo lahko različne vloge, glede na katere se razlikujejo zahteve po aplikacijskih programskih vmesnikih in storitvah v oblaku.

Naslednji seznam podaja tipe vlog in kategorije API-jev, ki se v posamezni vlogi uporabljajo [5]:

- **Razvijalec odjemalnih aplikacij:** razvijalec za končnega uporabnika razvija odjemalne aplikacije, ki temeljijo na oblaku. Uporablja se API-je za storitve v oblaku (kategorija 3)

- **Razvijalec aplikacij:** razvija tradicionalne aplikacije, ki uporabljajo oblak. Ti razvijalci uporabljajo običajne API-je (kategorija 1) kot tudi API-je za storitve v oblaku (kategorija 3)

- **Postavljalci:** njihova naloga je pakiranje, postavitev in vzdrževanje aplikacij, ki uporabljajo oblak. Prav tako skrbijo za upravljanje življenjskega cikla aplikacije. Uporablja se API-je za postavitev, storitve v oblaku in upravljanje slik virtualnih naprav (kategorije 2, 3 in 4)

- **Administratorji:** gre za delo z aplikacijami na večih nivojih, vključno z upravljanjem postavitve aplikacij in infrastrukture. Ti razvijalci uporabljajo API-je kategorije 2, 3 in 4.

- **Ponudniki oblakov:** njihovo delo je osredotočeno na infrastrukturo, ki leži pod storitvami v oblaku, ki jih ponujajo. Uporabljajo API-je kategorije 5.

2.6 Model zaračunavanja storitev

Najbolj značilen model zaračunavanja uporabe PaaS je na podlagi količine porabljenih virov (*Pay-As-You-Go*). Podobno kot pri storitvah elektrike in vode, uporabnik namreč za določeno obračunsko obdobje plača samo tiste vire, ki jih je uporabljal in to le toliko, kolikor jih je porabil. Veliko ponudnikov ponavadi omogoča nadzor in natančen vpogled nad porabo virov, kar pripomore k večji stroškovni učinkovitosti. Viri, ki se ponavadi zaračunavajo so naslednji:

- poraba računske moči na uro,
- prenosi podatkov v gigabajtih,
- število vhodno-izhodnih zahtev,
- velikost shrambe v gigabajtih,
- velikost prenosov podatkov v/iz shrambe v gigabajtih,
- število zahtev po dostopu do podatkov shrambe.

Poleg omenjenega modela se lahko uporablja tudi način zaračunavanja na podlagi naročnin in pa način, pri katerem uporabnik pri ponudniku vnaprej zakupi določeno količino izbranih zmogljivosti.

2.7 Trendi

Gartner je v svojih poročilih letošnje leto 2011 označil kot leto PaaS [3]. Večina večjih proizvajalcev programske opreme je namreč letos predstavila svojo rešitev na področju platforme kot storitve. Trenutno so posamezne rešitve zasnovane za bolj specifično problemsko področje, vendar Gartner napoveduje, da bodo nekje do leta 2015 postale celovite rešitve PaaS. Kljub popularnosti računalništva v oblaku Gartner modela PaaS v naslednjih petih letih ne vidi kot popolne zamenjave tradicionalnega razvoja aplikacij, ampak kot njegovo dopolnitev. Aplikacije na lokaciji in aplikacije v oblaku bodo soobstajale in se povezovale ter dopolnjevale med sabo.

2.7.1 Povezovalna platforma kot storitev

Korak k povezovanju aplikacij med seboj predstavlja iPaaS. Gartner je letos objavil poročilo, v katerem PaaS deli na aplikacijsko platformo kot storitev (A*pplication Platform as a Service*, *aPaaS*) in povezovalno platformo kot storitev (*Integration Platform as a Service*, *iPaaS*) [11]. V kategorijo aPaaS lahko uvrstimo vse, kar smo prej poimenovali pod PaaS, iPaaS pa je novost.

Kot že samo ime namiguje, je iPaaS platforma, ki ponuja množico storitev in funkcionalnosti za povezovanje aplikacij znotraj oblaka in za povezovanje aplikacij v oblaku ter tistih na lokaciji. Uporabniki iPaaS so isti kot pri PaaS, le storitve platforme so namenjene

povezovanju. Poleg ostalih lastnosti PaaS, naj bi rešitve iPaaS vključevale naslednje funkcionalnosti [11]:

- **Intermediacija** (*Intermediation*): ključno za iPaaS je sposobnost povezovanja raznih aplikacij in storitev, tako znotraj oblaka kot tudi med oblakom in na lokaciji.
- **Orkestracija** (*Orchestration*): gre za orkestracije med storitvami, kar vključuje preslikavo podatkov med storitvami, upravljanje delovnih tokov in procesiranje dogodkov.
- **Vsebnik storitev** (*Service Container*): uporabniki iPaaS morajo imeti možnost, da objavijo svoje storitve s pomočjo pristopov REST ali SOAP.
- **Varnost:** gre za sposobnost avtentikacije in avtorizacije dostopov do katerihkoli virov platforme, aplikacij v oblaku ter enkripcije in shranjevanja občutljivih podatkov.
- **Prehod za podatke organizacije** (*Enterprise Data Gateway*): iPaaS rešitve morajo zagotoviti varne kanale za dostop do podatkov na lokaciji kot tudi v oblaku. To pomeni, da mora biti prehod nameščen na lokaciji in delovati kot posrednik za dostop do podatkov organizacije in resursov kot so podatkovne baze, datotečni sistem, aplikacije in druge storitve.
- **Spremljanje v izvajalnem času** (*Runtime Monitoring*): tu gre za spremljanje aplikacije v skladu z dogovorom SLA.

Ker je model iPaaS še v začetni fazi razvoja, Gartner poudarja, da iPaaS rešitve, ki so trenutno na trgu, morda ne vključujejo zahtevanih funkcionalnosti. Gartner tako definira tri kategorije ponudnikov iPaaS, od katerih vsak poudarja drugo področje povezovanja [18]:

- povezovanje s pomočjo elektronskega poslovanja in B2B,
- povezovanje oblakov,
- uporaba dveh arhitekturnih modelov razvoja programske opreme, to je ESB in SOA.

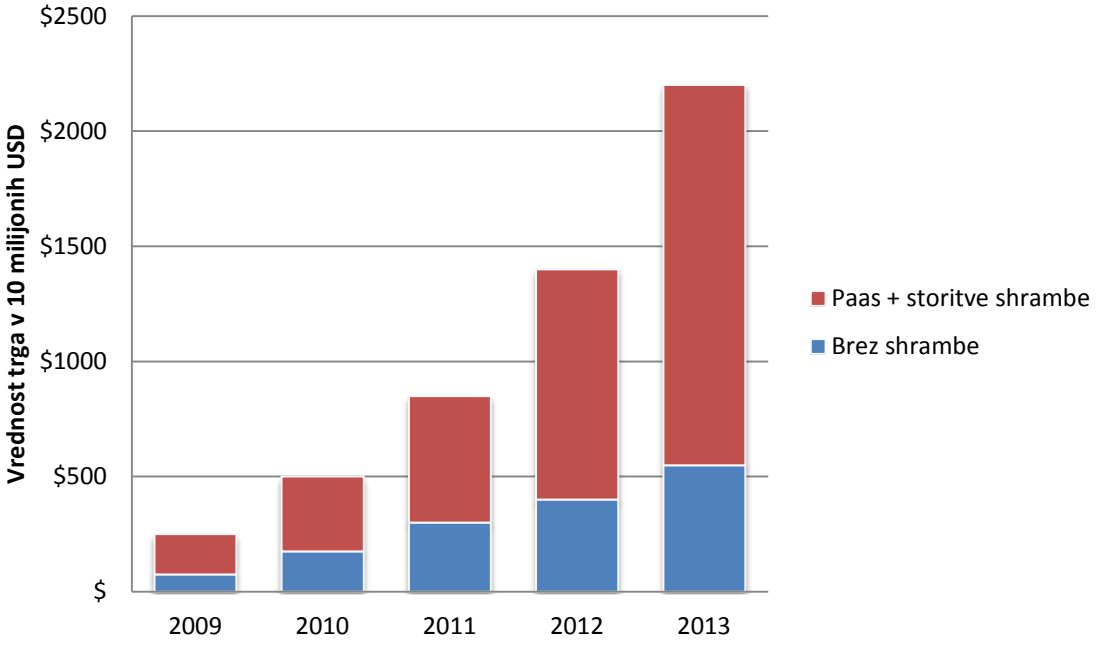
Glede na zahteve, ki jih ima posamezna organizacija, so nekatere rešitve boljše kot druge. Kratkoročno rešitev problema povezovanja predstavljata prvi dve kategoriji, elektronsko poslovanje in B2B te povezovanje oblakov, s katerima lahko hitro povežemo partnerske aplikacije in storitve v oblaku. Boljša in dolgoročno usmerjena rešitev pa naj bi bila po Gartnerju uporaba ESB in SOA.

Poglavje 3

Primerjava izbranih ponudnikov rešitev PaaS na trgu

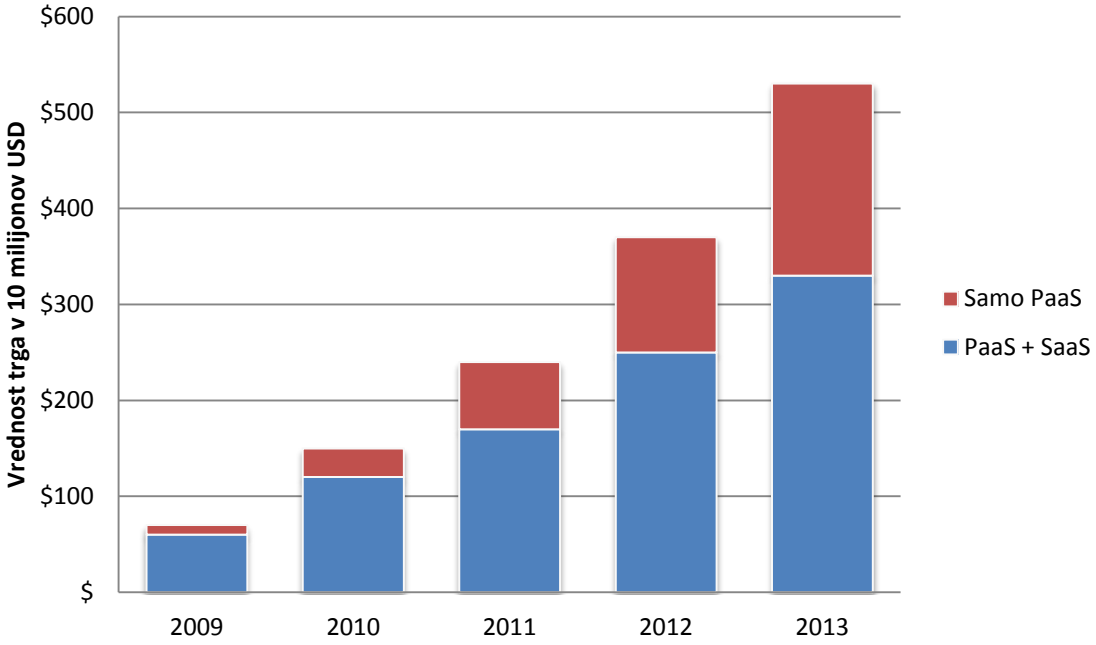
3.1 Stanje na trgu rešitev PaaS

Trg rešitev PaaS je kljub svoji trenutni majhnosti eden izmed najhitreje rastočih trgov. Ocenjena skupna letna stopnja rasti do leta 2013 je 68%. Vrednost trga tradicionalnih razvojnih platform, na katerega lahko rešitve PaaS potencialno posežejo, pa znaša več kot 50 milijard USD. Trend naraščanja prikazuje graf na sliki 3.1 [17]:



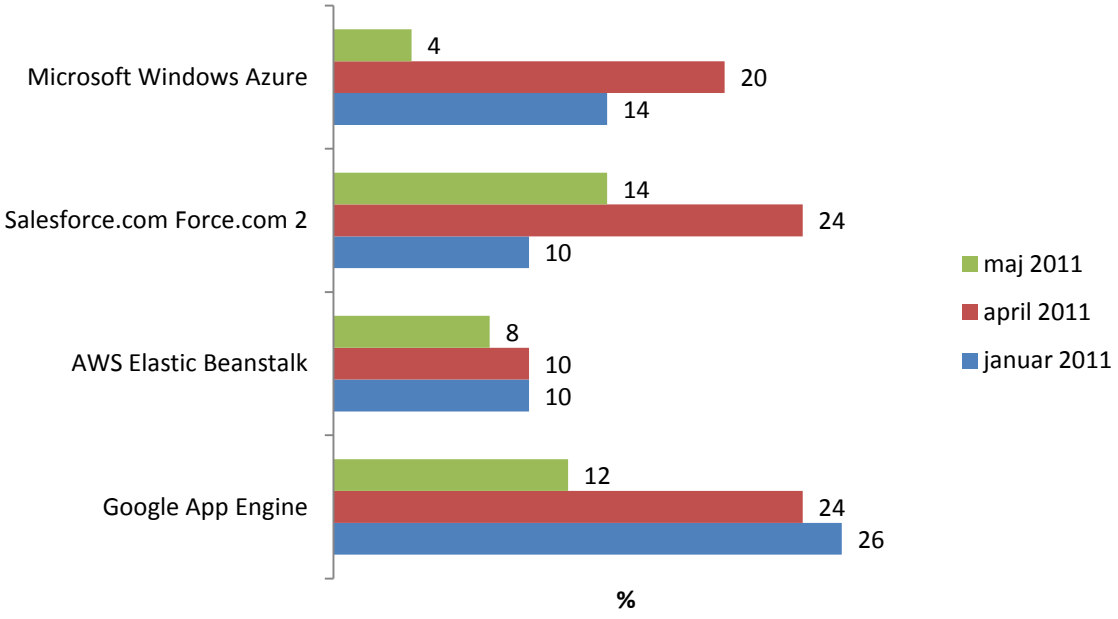
Slika 3.1. Trend naraščanja trga rešitev PaaS

Zaenkrat je uporaba rešitev PaaS v veliki večini (skoraj 75%) namenjena razvoju novih funkcionalnosti obstoječih SaaS aplikacij. Z vse večjo fleksibilnostjo rešitev PaaS in možnostmi uporabe širšega nabora programskih orodij in tehnologij, pa je pričakovati, da se bodo platforme PaaS vse bolj uporabljale tudi za razvoj novih aplikacij, kar je pravzaprav njihov glavni namen. Primerjavo med obema vrstama uporabe PaaS prikazuje graf na spodnji sliki 3.2 [17]:



Slika 3.2. Namen uporabe rešitev PaaS

Maja 2011 so trg rešitev PaaS obvladovali štirje ponudniki: Microsoft Windows Azure, Salesforce.com Force.com, AWS Elastic Beanstalk in Google App Engine. Napovedi analitske hiše Morgan Stanley so, da bodo ti štirje v prihodnjih letih nosilci razvoja rešitev PaaS in si bodo razdelili večinski delež trga [17]. Anketa, ki jo je analitska hiša opravila med izvršnimi direktorji informatike glede načrtovane uporabe rešitev PaaS v obdobju enega leta kaže, da je trend uporabe rešitev PaaS v naraščanju, na vrhu ponudnikov uporabljenih rešitev PaaS pa so že omenjeni štirje. Rezultate ankete prikazuje spodnja slika 3.3 [17]:



Slika 3.3. Rezultati ankete o načrtovanem prehodu na posamezno rešitev PaaS v enem letu

V nadaljevanju tega poglavja se bomo osredotočili na obravnavo izbranih ponudnikov rešitev PaaS: Microsoft Windows Azure, Salesforce.com Force.com, AWS Elastic Beanstalk in Google App Engine.

3.2 Microsoft Windows Azure

Microsoftovo platformo Windows Azure sestavlja infrastruktura virov strojne opreme, programske opreme, mreženja in shrambe, ki razvijalcem aplikacij nudijo sledeči nabor storitev [15]:

- Windows Azure,
- Windows Azure AppFabric,
- Microsoft SQL Azure,
- Windows Azure Marketplace DataMarket.

3.2.1 Storitve

3.2.1.1 Windows Azure

Windows Azure je operacijski sistem za storitve v oblaku in predstavlja temelj za izvajanje Windows aplikacij in shranjevanje podatkov v oblaku.

Storitve Windows Azure lahko razdelimo v sledeče sklope:

- **Računanje** (*Compute*): storitev računanja je zasnovana na temelju strežniškega operacijskega sistema Windows Server in služi izvajanju aplikacij. Te aplikacije so lahko razvite s pomočjo ogrodja .NET v programskih jezikih kot sta C# in Visual Basic, ali pa s pomočjo kakšnega drugega programskega jezika kot je Java. Razvijalci lahko uporabljajo Visual Studio ali kakšno drugo razvijalsko okolje in tehnologije kot so ASP.NET, Windows Communication Foundation (WCF) in skriptni jezik PHP.
- **Shramba** (*Storage*): ta storitev omogoča shranjevanje velikih binarnih objektov (*BLOB*), nudi sporočilne vrste za komunikacijo med komponentami Windows Azure aplikacij in nerelacijsko strukturo tabel s preprostim poizvedovalnim jezikom.
- **Krmilnik povezav** (*Fabric Controller*): Windows Azure teče na velikem številu naprav. Naloga krmilnika povezav je, da poveže te naprave znotraj Windows Azure podatkovnega centra v povezano celoto. Na vrhu te celote pa so zgrajene storitve računanja in shrambe.
- **Omrežje za dostavo vsebin** (*Content Delivery Network, CDN*): skrbi za hitrejši dostop do pogosto uporabljenih podatkov, tako da jih shranjuje na lokacijah, ki so fizično bližje uporabnikom.

- **Povezovanje** (*Connect*): omogoča aplikacijam na lokaciji povezovanje z aplikacijami v oblaku. Primer uporabe te storitve je lahko npr. Windows Azure aplikacija, ki se preko Windows Azure Connect povezuje na podatkovno bazo na lokaciji.

3.2.1.2 Microsoft SQL Azure

SQL Azure sestavljajo naslednje komponente:

- **Relacijska podatkovna baza** (*SQL Azure Database*): gre za sistem za upravljanje z relacijsko podatkovno bazo v oblaku. Ta tehnologija omogoča tako aplikacijam v oblaku kot tudi na lokaciji, da shranjujejo podatke na Microsoftove strežnike v Microsoftovih podatkovnih centrih.

- **Izdelava poročil** (*SQL Azure Reporting*): gre za komponento, ki se uporablja v navezavi z SQL Azure podatkovno bazo in na podlagi podatkov v oblaku omogoča kreiranje in objavljanje poročil.

- **Sinhronizacija podatkov** (*SQL Azure Data Sync*): ta komponenta omogoča sinhronizacijo podatkov med podatkovnimi bazami v oblaku ali pa med podatkovnimi bazami v oblaku in tistimi na lokaciji. Kodo za sinhronizacijo je prej razvijalec moral napisati s pomočjo ogrodja Microsoft Sync Framework, zdaj pa mora samo nastaviti določene nastavitve.

3.2.1.3 Windows Azure AppFabric

Storitev Windows Azure AppFabric vključuje naslednje komponente:

- **Storitveno vodilo** (*Service Bus*): služi za izpostavljanje storitev aplikacije drugim aplikacijam. Aplikacija izpostavi v oblak svoje končne točke (*endpoints*), preko katerih jo lahko uporabljajo ostale aplikacije v oblaku, kot tudi aplikacije, ki ne tečejo v oblaku.

- **Kontrola dostopa** (*Access Control*): kontrola dostopa nudi razvijalcu podporo za implementacijo mehanizmov prijavljanja kot sta Active Directory in Windows Live ID. Prav tako omogoča definiranje pravil za nadzor dostopa vsakega uporabnika na enem centralnem mestu.

- **Predpomnjenje** (*Caching*): aplikacije pogosto dostopajo do istih podatkov. Eden izmed načinov za pospeševanje dostopa do teh podatkov je njihovo predpomnjenje, s čimer se zmanjša število dostopov aplikacije do podatkovne baze.

3.2.1.4 Windows Azure Marketplace

Windows Azure Marketplace je storitev, ki uporabnikom omogoča iskanje in nakup aplikacij ter podatkov v oblaku.

Tržnico sestavljata dve komponenti:

- **Podatkovna tržnica** (*DataMarket*): omogoča iskanje in nakup podatkov tako uporabnikom kot aplikacijam. Z uporabo Windows Azure aplikacije Service Explorer lahko uporabnik vidi, kateri podatki so na voljo, nato pa se odloči za nakup. Aplikacije lahko do teh podatkov potem dostopajo preko zahtev REST ali protokola OData (*Open Data Protocol*), ki je spletni protokol za poizvedovanje in posodabljanje podatkov. Podatki, ki so na voljo na tržnici so lahko shranjeni na sami platformi z uporabo Windows Azure Storage ali SQL Azure Database, ali pa se nahajajo v podatkovnih centrih zunaj oblaka.

- **Tržnica aplikacij** (*AppMarket*): omogoča razvijalcem aplikacij v oblaku, da svoje aplikacije izpostavijo potencialnim strankam.

3.2.2 Varnost

Windows Azure ponuja sledeče varnostne mehanizme:

- **SLA:** je na voljo, dvostranska

- **Avtentikacija:** več načinov avtentikacije, npr. na podlagi uporabniškega imena in gesla, Windows Live ID, uporaba certifikatov, povezana identifikacija (*Federated Identity*, omogoča uporabniku enkratno identifikacijo na večih spletnih mestih)

- **Avtorizacija:** definiranje avtorizacijskih pravil

- **Integriteta:** zagotovljena s pomočjo avtentikacije in avtorizacije ter enkripcije (SSL, različni kriptografski mehanizmi)

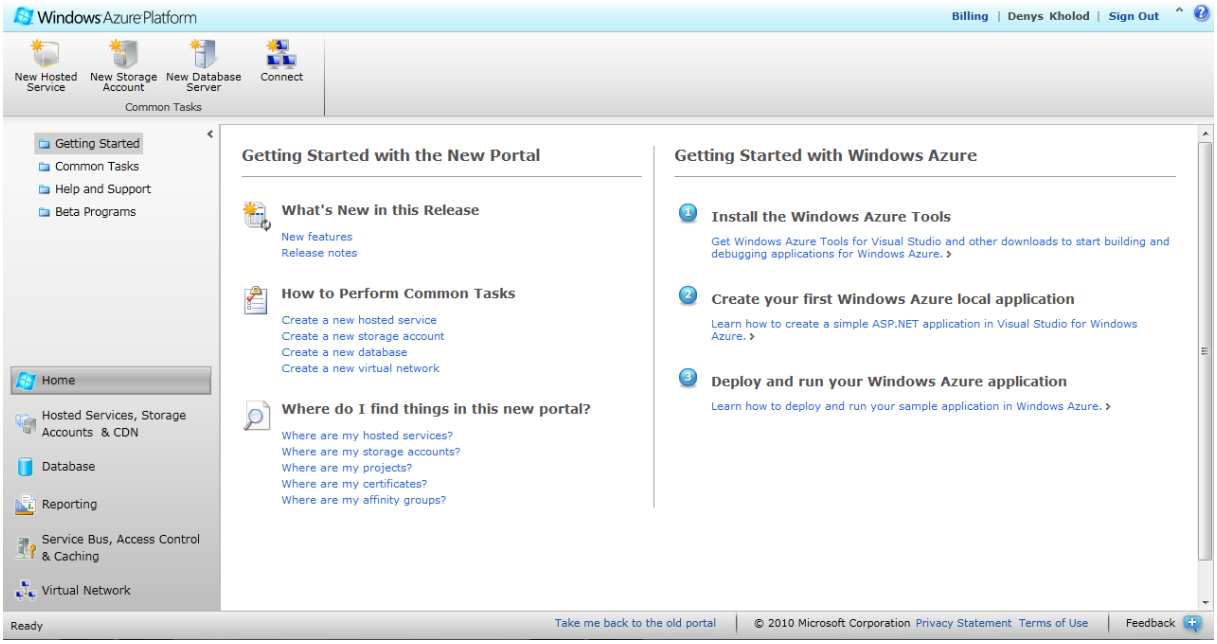
- **Zaupnost:** je definirana v okviru varnostne politike in zagotovljena s pomočjo avtentikacije, avtorizacije ter enkripcije

- **Razpoložljivost:** je zagotovljena v okviru SLA, različna za posamezne storitve (večinoma 99.9%)

3.2.3 Razvoj aplikacij

Razvoj Windows Azure aplikacije za razvijalce zgleda precej podobno kot razvoj .NET aplikacij. Kot pomoč pri razvoju ima Microsoft že na voljo projektne predloge v okolju Visual Studio. Prav tako Windows Azure SDK vsebuje verzijo Windows Azure okolja, ki lahko teče na razvijalčevem lokalnem računalniku. Imenuje se Windows Azure Development.

Razvijalec lahko razvije in razhroščuje svojo aplikacijo z uporabo lokalne simulacije in jo nato naloži na Windows Azure, ko je pripravljena za uporabo. Za upravljanje aplikacije na platformi ima razvijalec na voljo administracijsko konzolo, do katere dostopa s spletnim brskalnikom (slika 3.4.).



Slika 3.4. Administracijska konzola za Microsoft Windows Azure

Sama aplikacija je strukturirana kot ena ali več vlog. V izvajanju se tipično poganjata dve ali več instanc vsake vloge, vsaka instanca pa se izvaja kot samostojna virtualna naprava.

Vsaka Windows Azure aplikacija se lahko razvije z uporabo treh tipov vlog:

- **Spletne vloge** (*Web roles*): so namenjene predvsem izvajanju spletnih aplikacij. Znotraj vsake instance spletne vloge teče prednastavljen strežnik IIS 7 (*Internet Information Services*). Poleg Microsoftovih tehnologij kot sta ASP.NET in WCF (*Windows Communication Foundation*), je možno za razvoj aplikacij uporabiti tudi skriptni jezik PHP, Javo in druge tehnologije.

- **Delovne vloge** (*Worker roles*): te vloge so namenjene poganjanju različnim vrstam kode. Delovna vloga lahko poganja simulacijo, procesiranje videa itd. Aplikacija pogosto uporablja spletno vlogo za interakcijo z uporabniki, delovno vlogo pa za procesiranje opravil.

- **Vloge navideznih naprav** (*VM roles*): gre za vloge, ki lahko izvajajo navidezne slike strežnika Windows Server 2008 R2. Te navidezne slike poda uporabnik. Taka vloga je lahko uporabna v primeru selitve nekaterih strežniških Windows Server aplikacij, ki ne tečejo v oblaku, na Windows Azure.

Kadar razvijalec da na Windows Azure v izvajanje aplikacijo, posreduje tudi informacijo o nastavitvah. Med drugim ta informacija platformi pove, koliko instanc vsake vloge naj ustvari in požene. Krmilnik povezav nato kreira navidezno napravo za vsako instanco.

Windows Azure omogoča razvijalcem, da za instance vlog lahko izbirajo med več velikostmi navideznih naprav, ki imajo lahko različno število procesorskih jeder in različno velikost pomnilnika. Lastnik aplikacije lahko razbremeni aplikacijo, tako da poveča število instanc, ki se lahko izvajajo za eno ali več vlog. Krmilnik povezav bo v takem primeru zagnal nove navidezne naprave za te instance. Možno je tudi eksplicitno zmanjšati število instanc vloge, tako da se lahko aplikacija širi in krči v skladu z bremenom.

Implementacija večnajemniškega modela
Za implementacijo večnajemniškega modela Windows Azure lahko uporablja vse tri pristope deljenja vmesnega sloja programske opreme.

3.2.4 Model zaračunavanja storitev

Microsoft ponuja uporabnikom dva načina zaračunavanja storitev svoje platforme:

- naročnine, kjer naročnik zakupi določeno količino virov,
- model zaračunavanja na podlagi porabe (*Pay-As-You-Go*), pri katerem naročnik glede na cenik plača toliko, kolikor virov porabi.

Možna je tudi uporaba 90-dnevne brezplačne različice, ki ponuja količinsko omejeno uporabo virov.

3.3 Salesforce.com Force.com

Rešitev PaaS Force.com, ki jo je ponudil Salesforce.com, je bila prva razvojna platforma v oblaku na svetu. V zadnjem času jo je podjetje nadgradilo v Force.com 2. Gre pravzaprav za paket večih platform, vsaka pa je namenjena določeni vrsti uporabe. Platforme so sledeče [16,22]:

- **Appforce:** gre za originalno Force.com platformo, ki uporablja lasten programski jezik 4. generacije, imenovan Apex. Primarno je namenjena manjšim IT oddelkom.
- **Siteforce:** gre za razvojno okolje, ki je specializirano za razvoj spletnih strani brez pisanja kode.
- **VMforce:** gre za platformo, ki je rezultat sodelovanja s podjetjem Vmware. Omogoča poganjanje Java aplikacij na Force.com platformi, razvijalci pa lahko uporabljajo tudi javanska ogrodja kot je npr. Spring in razvojno okolje Eclipse.
- **Heroku:** podobno kot VMforce, Heroku razvijalcem omogoča razvoj in poganjanje aplikacij, napisanih v jeziku Ruby.
- **ISVforce:** gre za platformo, ki ponuja storitve in orodja, ki omogočajo prenos aplikacij na lokaciji v oblak. Gre za storitve postavitve, zaračunavanja, povezovanja na AppExchange in spremljanja aplikacij.
- **AppExchange:** gre za tržnico aplikacij, ki jih lahko naložimo in prilagodimo za svoje potrebe.
- **Database.com:** gre za storitev podatkovne baze v oblaku.

3.3.1 Storitve

Osredotočili se bomo na opis storitev platforme Appforce, oziroma Force.com.

3.3.1.1 Podatkovna baza

Razvijalcem je na voljo podatkovna baza, ki se razlikuje od tradicionalnih relacijskih podatkovnih baz. Podatki so namesto v tabelah shranjeni kot zapisi znotraj objektov. Vsak objekt je sestavljen iz določenega števila polj (*Fields*), ki so lahko sledečih tipov:

- **Identifikacijska polja** (*Identity Fields*): gre za polja, ki ustrezajo konceptu primarnega ključa v relacijskih podatkovnih bazah in se samodejno kreirajo in upravljajo za vsak objekt.

- **Sistemska polja** (*System fields*): gre za polja, iz katerih je možno podatke samo prebrati. Taka polja so npr. datum in čas kreiranja objekta, ID uporabnika, ki je kreiral objekt, itd.

- **Imensko polje** (*Name field*): gre za polje, ki predstavlja uporabniku razumljiv identifikator zapisa, ki je lahko tekst ali pa samodejno generirana številka.

Namesto primarnih in tujih ključev, na katerih temeljijo relacije v relacijski podatkovni bazi, se v podatkovni bazi Force.com uporabljajo relacijska polja (*Relational Fields*).

Objekti so v Force.com podatkovni bazi dejansko shranjeni kot metapodatki. V času izvajanja je poseben pogon tisti, ki poskrbi, da se s pomočjo analiziranja teh metapodatkov kreirajo dejanski objekti.

Za opravljanje poizvedb, Force.com podpira dva poizvedovalna jezika, ki se uporabljata v kombinaciji z jezikom Apex:

- **Salesforce Object Query Language (SOQL)**: gre za objektni poizvedovalni jezik, ki je podoben SQL, vendar namesto stikov uporablja relacije.

- **Salesforce Object Search Language (SOSL)**: gre za preprost jezik za iskanje objektov.

Apex omogoča tudi operacije za vstavljanje, brisanje in posodabljanje podatkov v podatkovni bazi.

3.3.1.2 Povezovalne storitve

Zunanji dostopi do podatkov

Zunanje aplikacije lahko dostopajo do podatkov Force.com aplikacij s pomočjo dveh vmesnikov API: Force.com Web Services API in Force.com REST API. Web Services API je vmesnik za SOAP spletne storitve, REST API pa za REST spletne storitve, oba pa omogočata neposredno izvajanje operacij nad podatki (poizvedovanje, kreiranje, brisanje, posodabljanje).

Force.com podpira tudi nabore orodij, ki omogočajo povezovanje Force.com aplikacij in aplikacij, razvitih na drugih platformah PaaS:

- **Force.com Toolkit for Azure:** omogoča Windows Azure aplikacijam uporabo vmesnika Force.com Web Services API za dostop in manipulacijo podatkov Force.com aplikacij.

- **Force.com for Amazon Web Services:** omogoča Force.com aplikacijam uporabo storitve podatkovne baze Amazon S3 in storitve računanja Amazon EC2. S pomočjo slednje se lahko ustvari navidezna naprava, ki vsebuje izvajalno okolje za aplikacije v skriptnem jeziku PHP, ki lahko dostopajo do Force.com aplikacij.

- **Force.com for Google App Engine:** gre za Python knjižnico, ki omogoča Google App Engine aplikacijam uporabo vmesnika Force.com Web Services API za dostop do podatkov Force.com aplikacij.

Zunanji dostopi do poslovne logike
Zunanja (pa tudi Force.com aplikacija) lahko uporablja poslovno logiko neke Force.com aplikacije z uporabo storitev Apex Web Services. Te storitve omogočajo, da lahko preko njih izpostavimo Apex razrede kot varne spletne storitve.

Force.com klici in prepletene storitve
Force.com klic (*Callout*) je funkcionalnost, ki jo aplikacija lahko uporabi za izgradnjo prepletene storitve (*Mashup*) – uporabniškega vmesnika, ki združuje vsebino iz večih virov. Primer je Force.com aplikacija, ki uporablja prepleteno storitev za prikaz forme za vnos podatkov o nepremičninah skupaj s klicem, ki prikaže lokacijo naslova na zemljevidu s pomočjo podatkov, ki jih priskrbi neka oddaljena storitev.

Obveščanje zunanjih aplikacij
Za samodejno obveščanje zunanjih aplikacij in storitev o dogodkih, ki se odvijajo znotraj Force.com aplikacij, se uporabljajo izhodna sporočila.

3.3.1.3 Poslovna logika

Force.com platforma omogoča enostavno avtomatizacijo poslovnih procesov in zahtev podjetja. Pogon delovnega toka (*Workflow engine*) nudi splošne procesne komponente za večkratno uporabo, kot npr. kreiranje opravil, dodeljevanje zapisov, časovne akcije, itd. Force.com omogoča enostavno vgraditev teh komponent v logiko aplikacije.

Za večjo prilagodljivost imajo razvijalci na voljo unikaten programski jezik Apex Code, ki omogoča implementacijo poljubne funkcionalnosti in poslovne logike. Med drugim omogoča tudi operacije nad podatki v podatkovni bazi, kreiranje prožilcev, ki se samodejno izvedejo glede na operacije nad podatkovno bazo, ter dostop in klic zunanjih spletnih storitev.

Kodo, ki je napisana v jeziku Apex, Force.com prevede in shrani kot metapodatke. Ob klicu interpreter na podlagi metapodatkov prevede kodo in jo naloži v pomnilnik. Ob morebitnem ponovnem klicu iste kode tako ni potrebno ponovno prevajanje.

Force.com samodejno preveri vse dostope do podatkov znotraj Apex razreda, da prepreči kodo, ki se drugače med izvajanjem ne bi izvedla. Vzdržuje tudi informacijo o pripadajočih odvisnih objektih, da prepreči spremembe metapodatkov, ki bi povzročile napake v odvisnih aplikacijah.

Za preprečevanje zlorab ali nenamerne prisvojitve deljenih virov, se hkrati z izvedbo Apex izvaja tudi množica upravljalcev in omejevanje virov. Force.com podrobno spremlja izvajanje Apex skripte in omejuje čas procesiranja ter količino pomnilnika, ki ga skripta lahko porabi,

število poizvedb in število matematičnih operacij, ki jih lahko izvede, itd. Vse te omejitve pripomorejo k zaščiti skalabilnosti in zmogljivosti deljene platforme.

Sintaksa Apex jezika je podobna Javi, zato je razvoj Force.com aplikacij z njim relativno enostaven.

3.3.2 Varnost

Salesforce Force.com ponuja sledeče varnostne mehanizme:

- **SLA:** trenutno ni na voljo
- **Avtentikacija:** več načinov avtentikacije, npr. avtentikacija na podlagi uporabniškega imena in gesla, povezana identifikacija (*Federated Identity*) z uporabo SAML (*Security Assertion Markup* Language), možnost uporabe različnih drugih mehanizmov za avtentikacijo kot je LDAP (*LightWeight Directory Access Protocol*)
- **Avtorizacija:** uporaba SAML, definiranje avtorizacijskih pravil
- **Integriteta:** zagotovljena s pomočjo avtentikacije in avtorizacije ter enkripcije (SSL)
- **Zaupnost:** je definirana v okviru varnostne politike in zagotovljena s pomočjo avtentikacije, avtorizacije ter enkripcije
- **Razpoložljivost:** ponudnik navaja 99.9% razpoložljivost, ni SLA

3.3.3 Razvoj aplikacij

Metapodatki

Ključen koncept pri razvoju aplikacij za Force.com platformo so metapodatki, ki smo jih že večkrat omenili pri opisu storitev. Izvajalni pogoni uporabljajo metapodatke, da generirajo aplikacijske komponente. Na primer, ko razvijalec razvija novo aplikacijo in definira neko poljubno tabelo ali napiše neko kodo, Force.com ne kreira tabele v podatkovni bazi ali prevede kode. Namesto tega preprosto shrani metapodatke, ki jih njegov pogon tekom izvajanja uporablja za generiranje navideznih aplikacijskih komponent. Ko nekdo želi spremeniti ali prilagoditi aplikacijo, je potrebna samo posodobitev pripadajočih metapodatkov. Kadar uporabnik uporablja aplikacijo, izvajalni aplikacijski generator (*runtime application generator*) uporablja metapodatke za prikaz aplikacijskih komponent v uporabniškem vmesniku.

Ker so metapodatki ključna sestavina Force.com aplikacij, mora izvajalni pogon platforme optimizirati dostop do metapodatkov. V nasprotnem primeru bi pogosti dostopi do metapodatkov onemogočili skalabilnost platforme.

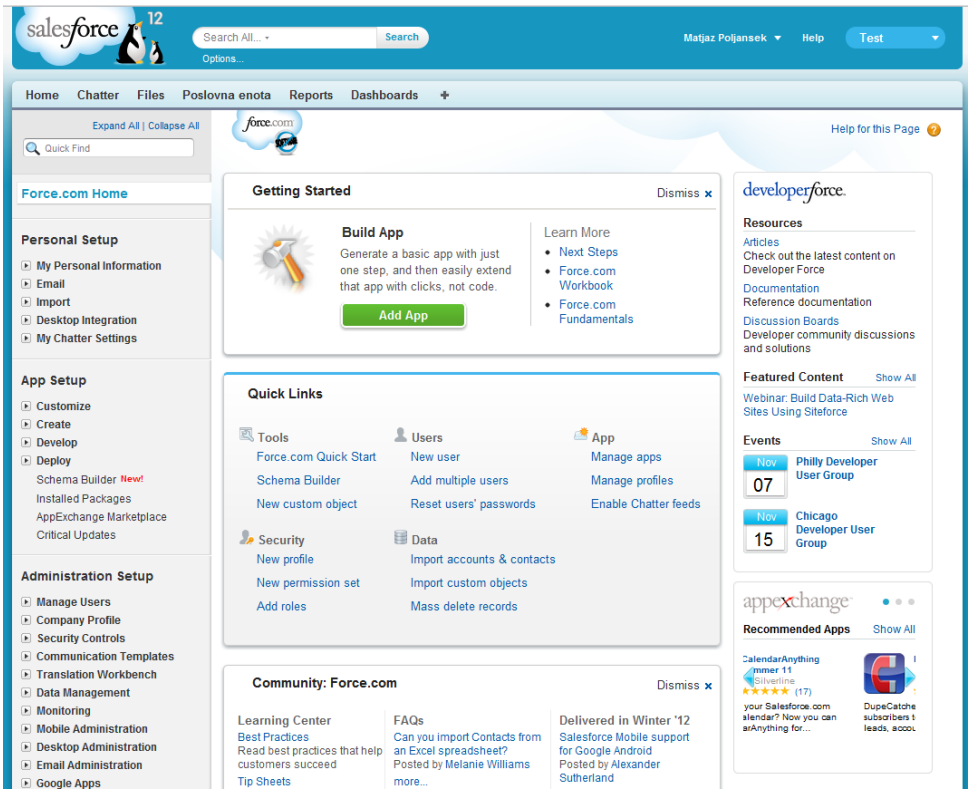
Okolja

Okolje (*Environment*) ali organizacija (*Organization*), kot ga poimenuje Salesforce, je posamezna instanca platforme Force.com, ki omogoča dostop, razvoj ali postavitev aplikacij z omejitvami in funkcionalnostmi, ki so odvisne od izbranega okolja.

Razvijalcu aplikacij sta na voljo dve okolji, in sicer:

- **Razvojno okolje** (*Development Environment*): gre za okolje, ki omogoča razvoj novih in razširitev obstoječih Force.com aplikacij. Razvoj aplikacij je možen prek spletnega brskalnika, lahko pa se za to uporablja lokalno razvojno okolje Force.com IDE, ki temelji na razvojnem okolju Eclipse in tako razvijalcem omogoča poznano okolje za pisanje kode, prevajanje, testiranje, pakiranje in postavitev aplikacij. Force.com IDE lahko deluje v povezavnem (*Online*) ali brezpovezavnem (*Offline*) načinu. V obeh primerih se s platformo Force.com izmenjujejo metapodatki preko vmesnika Metadata API, le da se pri brezpovezavnem metapodatki sinhronizirajo potem, ko je vzpostavljena povezava.
- **Testno okolje** (*Testing Environment*): gre za okolje, ki je namenjeno testiranju delovanja aplikacij, preden se jih postavi v produkcijsko okolje.

Spodnja slika 3.5 prikazuje administracijsko konzolo za kreiranje, postavitev in upravljanje aplikacije v razvojnem okolju:



Slika 3.5. Administracijska konzola za Force.com

Obstaja še tretje, **produkcijsko okolje** (*Production Environment*), ki pa je namenjeno končnim uporabnikom, npr. poslovnim organizacijam, saj se v njem nahajajo aktivni podatki in aplikacije, ki jih poslovne organizacije potrebujejo za svoje poslovanje.

Na voljo je tudi storitev Force.com Code Share, ki omogoča razvijalcem iz celega sveta, da sodelujejo na razvoju, testiranju in postavitvi aplikacij v oblak. Ker se Code Share integrira z večino sistemov za upravljanje z izvorno kodo, lahko razvijalci shranijo definicije svojih Force.com aplikacij v sistem za nadzor izvorne kode in postavijo te aplikacije v testno ali produkcijsko okolje.

Uporabniški vmesnik
Force.com nudi dve možnosti za kreiranje in prilagoditev uporabniškega vmesnika aplikacije Prva je Force.com Builder, ki preko enostavnega povleci-in-spusti vmesnika omogoča kreiranje in spreminjanje razporeditve ter zaporedja podatkovnih polj na straneh, preimenovanje in razporejanje zavihkov, kreiranje kompleksnih poročil in kreiranje različnih pogledov na podatke za različne uporabnike.

Za večji nadzor nad uporabniškim vmesnikom pa je na voljo Visualforce, ki je kompletno ogrodje za kreiranje in poganjanje praktično kateregakoli uporabniškega vmesnika, za katerokoli aplikacijo in na katerikoli napravi. Uporablja HTML in AJAX ter Flex za poslovne aplikacije.

Visualforce sestavljajo naslednji funkcionalni sklopi:

- **Strani** (*Pages*): omogočajo izdelavo načrta za uporabniški vmesnik aplikacije.
- **Komponente** (*Components*): nudijo možnost izdelave novih aplikacij, ki samodejno ustrezajo izgledu Salesforce.com aplikacij ali pa enostavno prilagajajo in razširjajo Salesforce.com uporabniški vmesnik določenim zahtevam..
- **Krmilniki logike** (*Logic Controllers*): krmilnik omogoča uporabniku, da definira obnašanje uporabniškega vmesnika.

Implementacija večnajemniškega modela
Za implementacijo večnajemniškega modela Force.com uporablja pristop 1: deljenje vmesnega sloja programske opreme z eno samo instanco aplikacije.

3.3.4 Model zaračunavanja storitev

Salesforce zaračunava uporabo svoje Force.com platforme mesečno, glede na število uporabnikov, pri čemer imamo na voljo tri različne pakete, ki se razlikujejo po ceni naročnine, številu postavljenih aplikacij in omejitvah glede uporabe virov. Možen je tudi 30-dnevni brezplačni preizkus vsakega izmed treh paketov.

3.4 Amazon Web Services Elastic Beanstalk

Amazon je eden največjih ponudnikov storitev v oblaku, ki se je do nedavnega s svojo množico storitev v oblaku Amazon Web Services (AWS) uveljavljal predvsem kot ponudnik IaaS. Januarja 2011 pa je vstopil tudi na trg PaaS in tako na temeljih obstoječih storitev v oblaku ponudil rešitev AWS Elastic Beanstalk [1].

3.4.1 Storitve

Kot smo že omenili, Amazon omogoča veliko storitev v oblaku pod skupnim imenom AWS, ki jih lahko uporabljajo tudi razvijalci na AWS Elastic Beanstalk. V naslednjih podpoglavjih bomo opisali nekatere pomembnejše med njimi.

3.4.1.1 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (Amazon EC2) je storitev računanja v oblaku. Razvijalcu omogoča, da kreira navidezne naprave, na katerih se izvajajo njegove aplikacije. Možno je izbirati med različnimi konfiguracijami navideznih naprav, med drugim lahko razvijalec izbere operacijski sistem, ki se bo izvajal na navidezni napravi, podatkovno bazo in različne zmogljivosti računalniških virov (procesorska moč, pomnilnik, itd.).

Amazon EC2 omogoča samodejno skalabilnost aplikacij, in sicer glede na pogoje, ki jih postavi razvijalec. Npr., če je obremenjenost aplikcije takšna, da instance navideznih naprav v povprečju uporabljajo več kot 70% procerske moči, potem se lahko samodejno kreirajo dodatne 3 instance. Podobno se lahko število instanc samodejno zmanjša, ko je obremenjenost aplikacije majhna.

Razvijalec lahko posamezne instance navideznih naprav postavi na eno izmed šestih geografskih regij. Vsaka regija vsebuje več razpoložljivostnih območij (*Availability zones*), na katere se instance navideznih naprav replicirajo. Gre za geografsko razpršene lokacije, ki so zgrajene z namenom, da preprečijo odpoved delovanja aplikacij v primeru, da pride do napak na posamezni lokaciji.

3.4.1.2 Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) je storitev, ki razvijalcem omogoča shranjevanje podatkov za potrebe njihovih spletnih aplikacij, ki zahtevajo visoko skalabilnost.

Podatki so shranjeni kot objekti, vsak objekt pa lahko vsebuje 1 bajt pa vse do 5 terabajtov podatkov. Število shranjenih objektov je neomejeno. Vsak objekt je shranjen v t.i. vedru (*Bucket*), do njega pa se dostopa s pomočjo unikatnega ključa, ki se dodeli razvijalcu. Storitev omogoča izvajanje preprostih operacij, kot so branje, pisanje in brisanje objektov. Razvijalec ima možnost izbire ene izmed šestih geografskih regij, na kateri želi, da je vedro fizično

shranjeno (za Evropo je to Irska). To omogoča zmanjšanje zakasnitve pri prenosu podatkov. Storitev zagotavlja, da objekti, ki so v okviru vedra shranjeni v neki regiji, nikoli ne zapustijo te regije, razen če to eksplicitno želimo.

Objekti so lahko zasebni ali javni, za specifične uporabnike pa je možno definirati različna dovoljenja. Za zaščito pred nedovoljenim dostopom so na voljo različni avtentikacijski mehanizmi, prav tako je zagotovljena enkripcija podatkov za varno prenašanje in nalaganje.

3.4.1.3 Amazon SimpleDB

Amazon SimpleDB je visoko razpoložljiva in skalabilna nerelacijska podatkovna baza. Odpravlja potrebo po administratorjih, saj sama poskrbi za vzdrževanje strojne in programske opreme, replikacijo in indeksiranje podatkov ter optimizirano delovanje. Razvijalec uporablja samo operaciji za shranjevanje in poizvedovanje podatkov, za vse ostalo poskrbi podatkovna baza.

Za razliko od relacijskih podatkovnih baz, Amazon SimpleDB omogoča prilagodljiv podatkovni model, ki se ne podreja vnaprej definirani shemi. To omogoča enostavno dodajanje novih podatkov, ki se tudi samodejno indeksirajo. Storitev se lahko uporablja tudi v navezi z Amazon S3, npr. za shranjevanje kazalcev na objekte, ki so shranjeni v S3.

3.4.1.4 Amazon Relational Database Service

Amazon Relational Database Service (Amazon RDS) je storitev, ki omogoča relacijsko podatkovno bazo. Podobno kot pri Amazon SimpleDB, je tudi v tem primeru razvijalec razbremenjen administratorskih opravil.

Razvijalec lahko preko administracijske konzole ali vmesnika API ustvari instanco relacijske podatkovne baze, pri čemer lahko izbira med podatkovnima bazama MySQL in Oracle. Ker gre za relacijsko podatkovno bazo, lahko razvijalci svoje obstoječe aplikacije na lokaciji lažje prenesejo v oblak. Prav tako lahko do podatkovne baze dostopajo s poljubnim orodjem, ki so ga uporabljali prej.

3.4.1.5 Amazon Simple Queue Service

Amazon Simple Queue Service (Amazon SQS) je storitev, ki razvijalcem omogoča sporočilne vrste za zanesljivo shranjevanje sporočil, ki potujejo med komponentami njihovih aplikacij. Te komponente so lahko med seboj neodvisne in porazdeljene po omrežju, prav tako storitev ne zahteva, da so vedno razpoložljive.

Razvijalci lahko na eni izmed šestih geografskih regij kreirajo neomejeno število sporočilnih vrst Amazon SQS, ki lahko vsebujejo neomejeno število sporočil. Sporočila se lahko pošiljajo in berejo istočasno, v vrsti pa se lahko zadržujejo do 14 dni. Storitev omogoča tudi deljeno uporabo neke instance sporočilne vrste z drugimi uporabniki storitev AWS.

3.4.1.6 Amazon Elastic MapReduce

Amazon Elastic MapReduce je storitev, ki je namenjena predvsem enostavnemu in stroškovno učinkovitemu procesiranju velikih količin podatkov. Uporablja Apache-jevo implementacijo Google-ovega ogrodja MapReduce, imenovano Hadoop, s pomočjo katerega lahko razvijalci razvijajo aplikacije, ki opravljajo podatkovno intenzivna opravila kot so indeksiranje spletnih strani, podatkovno rudarjenje, analiza dnevniških datotek, strojno učenje, finančne analize, itd.

Ogrodje MapReduce oziroma njegova implementacija Hadoop, temelji na dveh operacijah: *map* in *reduce*. Velike količine podatkov se najprej razbijejo na manjše dele, ki se nato lahko vzporedno procesirajo s pomočjo funkcije *map*. Funkcija *reduce* pa nato združi vmesne rezultate procesiranja, ki so izhod funkcije *map,* v končen rezultat.

Za vir podatkov, ki se analizirajo s pomočjo Amazon Elastic MapReduce in za shranjevanje končnih rezultatov, se uporablja storitev podatkovne baze Amazon S3.

3.4.1.7 Amazon Web Services Identity and Access Management

Amazon Web Services Identity and Access Management (IAM) je storitev, ki je namenjena nadzoru in upravljanju varnosti pri dostopu do storitev in virov AWS. Omogoča kreiranje in upravljanje uporabnikov storitev AWS, ter različne mehanizme za avtentikacijo (uporabniško ime in geslo, povezana identifikacija, itd.) ter avtorizacijo (dovoljenja za dostop do omejenih storitev in virov AWS, omejitev dostopa na podlagi časa, itd.).

3.4.2 Varnost

AWS Elastic Beanstalk ponuja sledeče varnostne mehanizme:

- **SLA:** opredeljen samo za nekatere storitve, dvostranski
- **Avtentikacija:** več načinov avtentikacije, ki jih nudi storitev IAM (avtentikacija na podlagi uporabniškega imena in gesla, uporaba generiranih parov ključev, varnostnih žetonov, povezane identifikacije (*Federated Identity*), itd.)
- **Avtorizacija:** več načinov avtorizacije, ki jih nudi storitev IAM (npr. definiranje varnostnih skupin (*Security Groups*)
- **Integriteta:** zagotovljena s pomočjo avtentikacije in avtorizacije ter enkripcije (SSL)
- **Zaupnost:** je definirana v okviru varnostne politike in zagotovljena s pomočjo avtentikacije, avtorizacije ter enkripcije
- **Razpoložljivost:** je podana v okviru SLA (večinoma 99.9%), vendar ne za vse storitve

3.4.3 Razvoj aplikacij

AWS Elastic Beanstalk trenutno podpira samo razvoj aplikacij v programskem jeziku Java, v delu pa je tudi podpora za druge programske jezike. Aplikacija lahko uporablja servlete (*Servlets*) in tehnologijo JavaServer Pages. Možno je uporabiti tudi vse Java EE razrede in API-je. Za dostop do množice storitev AWS je na voljo Java AWS SDK.

Preden razvijalec naloži verzijo aplikacije (npr. Java WAR datoteko) na AWS Elastic Beanstalk, mora podati tip vsebnika (*Container*). Vsaka aplikacija se namreč izvaja v vsebniku, ki določa infrastrukturo in programsko opremo, ki se bo uporabljala za izvajalno okolje aplikacije. Privzeti vsebnik omogoča izvajanje Java aplikacij v okolju Tomcat 7, ki uporablja Amazonovo različico operacijskega sistema Linux, spletni strežnik Apache in aplikacijski strežnik Apache Tomcat. Vsebnik lahko naenkrat vsebuje samo eno verzijo aplikacije, lahko pa se istočasno izvaja več istih verzij ali pa različne verzije aplikacije.

AWS Elastic Beanstalk za vsebnik kreira vire, ki so potrebni za izvajanje aplikacije. Ti vključujejo en izravnalnik bremena in skupino za samodejno skalabilnost (*Auto Scaling*), ki jo sestavlja ena ali več Amazon EC2 instanc.

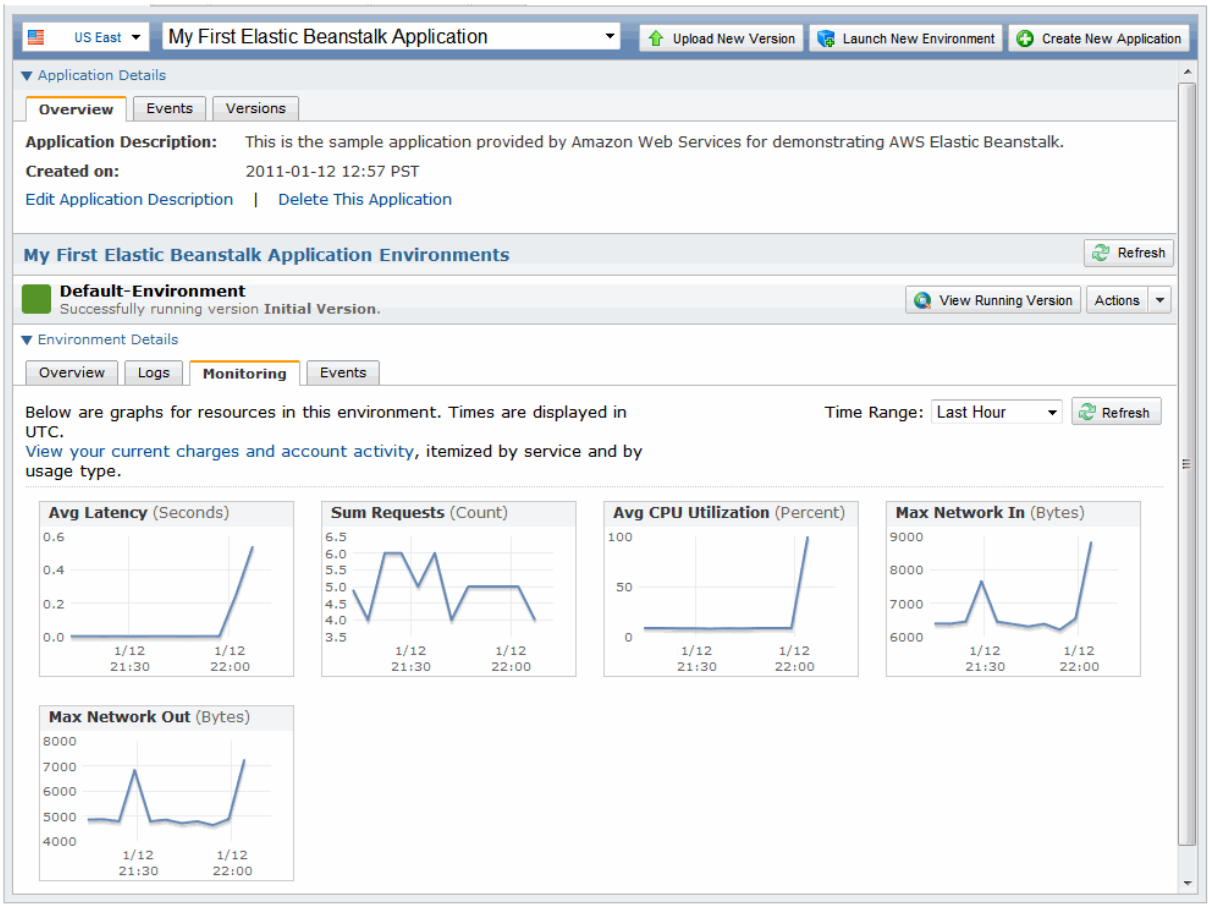
Ko se aplikacija kreira in postavi, so informacije o aplikaciji, kot so razne metrike, dogodki in stanja okolja vidni preko API-jev, ukazne vrstice in administracijske konzole AWS Management Console (slika 3.6).

Aplikacijo lahko kreiramo na enega izmed štirih različnih načinov:

- zavihek AWS Elastic Beanstalk v konzoli AWS Management Console,
- orodja v ukazni vrstici,
- nabor orodij AWS Toolkit za razvojno okolje Eclipse,
- API-ji spletnih storitev.

Implementacija večnajemniškega modela

Za implementacijo večnajemniškega modela AWS Elastic Beanstalk uporablja pristop 4: virtualizacijo.



Slika 3.6. Administracijska konzola za AWS Elastic Beanstalk

3.4.4 Model zaračunavanja storitev

Amazon uporabo platforme Elastic Beanstalk zaračunava glede na količino porabljenih virov (*Pay As You Go*). Možna je tudi enoletna brezplačna uporaba platforme, ki zajema količinsko omejeno uporabo virov.

3.5 Google App Engine

Google App Engine je Googlova rešitev na področju PaaS. Omogoča razvoj in gostovanje spletnih aplikacij na isti infrastrukturi, na kateri tečejo Googlove aplikacije (npr. Gmail) [4].

3.5.1 Storitve

3.5.1.1 Podatkovna baza

Google App Engine nudi storitev za shranjevanje podatkov Datastore, ki je ključna za skalabilnost Google App Engine aplikacij. Uporabniku omogoča dve možnosti shrambe podatkov, ki se razlikujeta v zahtevah glede zanesljivosti in konsistenčnosti. Storitev shranjevanja podatkov bomo bolj podrobno opisali, saj jo bomo uporabili pri razvoju in gostovanju naše spletne aplikacije na platformi App Engine.

Visoko replikacijska shramba
Visoko replikacijska shramba (*High Replication Datastore*) je privzeta shramba pri izdelavi nove aplikacije. Je visoko razpoložljiva in zanesljiva rešitev za shranjevanje podatkov. Branja in pisanja se lahko izvajajo tudi med časom izpada in je izjemno odporna tudi v primeru katastrofalnih odpovedi. Je najdražja opcija.

Shramba tipa gospodar/suženj
Shramba gospodar / suženj (*Master/Slave Datastore*) asinhrono replicira podatke na druge podatkovne centre, medtem ko se zapisujejo. V določenem času je gospodar za pisanje samo en podatkovni center, kar posledično na eni strani zagotavlja močno konsistenčnost za vsa branja in poizvedbe, po drugi strani pa to pomeni, da podatki v primeru napak ali planirane odpovedi nekaj časa ne bodo na voljo. Ta način shrambe je priporočljiv v primeru, da ne potrebujemo visoke razpoložljivosti podatkov, lahko toleriramo občasne zakasnitve pri dostopanju do podatkov in imamo omejena finančna sredstva.

BigTable
Storitev za shrambo podatkov Datastore ne temelji na tradicionalni, relacijski podatkovni bazi, ampak je osnovana na podlagi Googlove lastne nerelacijske podatkovne baze BigTable. BigTable je visoko porazdeljena in skalabilna podatkovna baza, namenjena shranjevanju ogromnih količin podatkov, porazdeljenih po več tisoč strežnikih.

Podatkovni model BigTable
BigTable si lahko predstavljamo kot objektno podatkovno bazo. Podatkovni objekt se imenuje entiteta (*Entity*). Vsaka entiteta je določena z enoličnim identifikatorjem ali ključem. Ključ je sestavljen iz podatka o tipu entitete (*Kind*), imenu entitete (lahko je določeno v okviru aplikacije, ali pa ga podatkovna baza generira sama), vsebuje pa lahko tudi pot do nadrejene entitete. Več entitet istega tipa lahko združimo v grupe (*Entity groups*), ki omogočajo, da v

okviru ene transakcije izvedemo operacije nad večimi entitetami. Vsaka entitetna grupa ima ključ starša (*Parent key*), ki enolično določa celotno entitetno grupo.

Entiteta ima lahko eno ali več lastnosti, ki so lahko različnih podatkovnih tipov. Za razliko od relacijskega podatkovnega modela ni nujno, da imata dve entiteti istega tipa iste lastnosti ali isti tip vrednosti za isto lastnost.

Aplikacija lahko do entitete v podatkovni bazi dostopa na podlagi njenega ključa, ali pa izvede poizvedbo. Aplikacija ne more dostopati do entitet, ki pripadajo drugim aplikacijam, ampak samo do tistih, ki jih sama kreira.

Operacije nad podatkovno bazo

Za dostop in koriščenje storitve Datastore imamo na voljo dva vmesnika API: standardni in nizko-nivojski.

Uporaba standardnega, višjenivojskega API-ja (kot sta npr. *Java Data Objects* (*JDO*) in *Java Persistence API* (*JPA*) za Javo) omogoča prenosljivost aplikacij na druge ponudnike storitev v oblaku in uporabo drugih tehnologij za podatkovne baze.

V kolikor je razvijalec prepričan, da bo aplikacija tekla le na Google App Engine platformi, pa se lahko poslužuje nizkonivojskega API-ja. Ta omogoča samo preproste operacije nad entitetami, kot so branje (*get*), pisanje (*put*), brisanje (*delete*) in poizvedovanje (*query*).

3.5.1.2 Ostale storitve

- **Google Cloud SQL (v razvoju):** Gre za storitev, ki razvijalcem omogoča, da za shranjevanje podatkov uporabljajo tradicionalno relacijsko podatkovno bazo MySQL in poizvedovalni jezik SQL. Prednost te storitve je večja prenosljivost podatkov, saj lahko podatke iz obstoječe baze MySQL enostavno prenesemo na Google App Engine, lahko pa jih iz Google App Engine tudi izvozimo. Aplikacije v Javi lahko storitev uporabljajo s pomočjo vmesnika JDBC (*Java DataBase Connectivity*), aplikacije v Pythonu pa imajo za ta namen na voljo vmesnik DB-API. Storitev je zaenkrat še v razvoju in je na voljo omejenemu številu razvijalcev.

- **Google BigQuery:** BigQuery je storitev, ki je namenjena hitremu izvajanju poizvedb SQL nad ogromnimi količinami podatkov, ki lahko zavzamejo več milijard vrstic. Podatki, nad katerimi izvajamo poizvedbe, se uvozijo v obliki CSV datoteke in se shranijo v obliki tabele. Podprti podatkovni tipi so nizi, številke in boolean vrednosti. Možno je izvajati samo stavke za poizvedovanje (SELECT). Storitev je zaenkrat na voljo samo omejenemu številu razvijalcev.

- **Google Cloud Storage (eksperiment, samo za Python):** Google Cloud Storage je storitev, ki omogoča shranjevanje velikih datotek kot objekte, pri čemer je shranjen objekt lahko poljubno velik. Omogoča tudi nekatere dodatne funkcionalnosti, ki jih storitev Datastore ne nudi, in sicer so to seznami za nadzor dostopa (*Access Control Lists, ACLs*), možnost avtentikacije in avtorizacije z OAuth 2.0. ter vmesnik API, ki temelji na standardu REST. Storitev je zaenkrat eksperimentalna in na voljo samo za aplikacije razvite v Pythonu.

- **Prospective Search:** Prospective Search je storitev, ki omogoča realnočasovno spremljanje podatkov, ki prihajajo v našo aplikacijo. Razvijalec mora predhodno definirati poizvedbe, ki se lahko tekom izvajanja aplikacije istočasno izvedejo nad podatki, ki prihajajo v aplikacijo. Primer uporabe take storitve so aplikacije, ki spremljajo objavljene komentarje v družabnih omrežjih kot je Facebook.

- **MapReduce (eksperiment, samo za Python):** MapReduce je model računanja, ki ga je Google razvil za potrebe učinkovitih analiz velikih količin podatkov. Omenili smo ga že v okviru storitve Amazon Elastic MapReduce, ki jo nudi platforma AWS Elastic Beanstalk. Za App Engine je zaenkrat na voljo kot eksperiment in sicer samo za aplikacije v Pythonu.

- **App Identity:** omogoča identifikacijo aplikacije, da se lahko predstavi zunanjim sistemom.

- **Blobstore:** omogoča aplikacijam strežbo velikih podatkovnih objektov (*BLOB*).

- **Capabilities:** gre za storitev, ki aplikaciji omogoča, da detektira izpade in načrtovane prekinitve delovanja za specifične storitve. S pomočjo te informacije lahko onemogočimo nerazpoložljive storitve v aplikaciji, preden izpad občutijo uporabniki.

- **Channel:** omogoča kreiranje obstojne povezave med aplikacijo in Googlovimi strežniki in tako aplikaciji omogoči pošiljanje sporočil odjemalcem v realnem času in brez povpraševanja. To je uporabno za aplikacije, ki so namenjene takojšnjemu obveščanju uporabnikov (npr. klepetalnice, igre...).
- **Images:** omogoča manipulacijo s slikami (spreminjanje velikosti, obračanje, rezanje,..).
- **Mail:** omogoča aplikacijam pošiljanje in sprejemanje e-poštnih sporočil, ter sprejemanje sporočil od App Engine v obliki HTTP zahtev.
- **Memcache:** omogoča storitev predpomnjenja za visoko skalabilne in zmogljive aplikacije, ki za določena opravila uporabljajo podatke v predpomnilniku.
- **Multitenancy:** omogoča aplikacijam implementacijo večnajemniškega modela.
- **Oauth:** omogoča aplikacijam uporabo OAuth protokola. Ta uporabniku omogoča, da tretji osebi brez posredovanja svojega uporabniškega imena in gesla omogoči omejene pravice za dostop do spletne aplikacije v njegovem imenu. Tretja oseba je lahko spletna aplikacija ali katerakoli aplikacija, ki omogoča uporabniku priklic spletnega brskalnika.
- **Task Queues:** omogoča aplikacijam, da delo organizirajo v opravila in jih postavijo v vrsto.
- **URL Fetch:** omogoča aplikacijam pridobivanje resursov in komuniciranje z drugimi gostitelji preko spleta s pošiljanjem HTTP in HTTPS zahtev.
- **Users:** omogoča aplikacijam storitve avtentikacije s pomočjo Google Accounts, OpenID identifikacije ali lastnih domen.
- **XMPP:** omogoča aplikacijam takojšnje pošiljanje in sprejemanje sporočil od in k uporabnikom, ki uporabljajo XMPP združljive storitve za takojšnje sporočanje (npr. Google Talk).

3.5.2 Varnost

Google App Engine ponuja sledeče varnostne mehanizme:

- **SLA:** je na voljo, dvostranski
- **Avtentikacija:** uporabniško ime in geslo (Google Accounts), povezana identifikacija z uporabo OpenID, možnost uporabe lastnega ogrodja za avtentikacijo
- **Avtorizacija:** uporabniške vloge (samo vlogi »navadni uporabnik« ali »administrator«, definiranja poljubnih uporabniških vlog App Engine ne podpira), možnost uporabe odprtega standarda za avtorizacijo OAuth (*Open Authorization*)

- **Integriteta:** zagotovljena s pomočjo avtentikacije in avtorizacije ter enkripcije (SSL)
- **Zaupnost:** je definirana v okviru varnostne politike in zagotovljena s pomočjo avtentikacije, avtorizacije ter enkripcije
- **Razpoložljivost:** ponudnik v okviru SLA zagotavlja razpoložljivost 99.95%

3.5.3 Razvoj aplikacij

Google App Engine trenutno podpira razvoj aplikacij v programskih jezikih Python, Java in lastnem programskem jeziku Go.

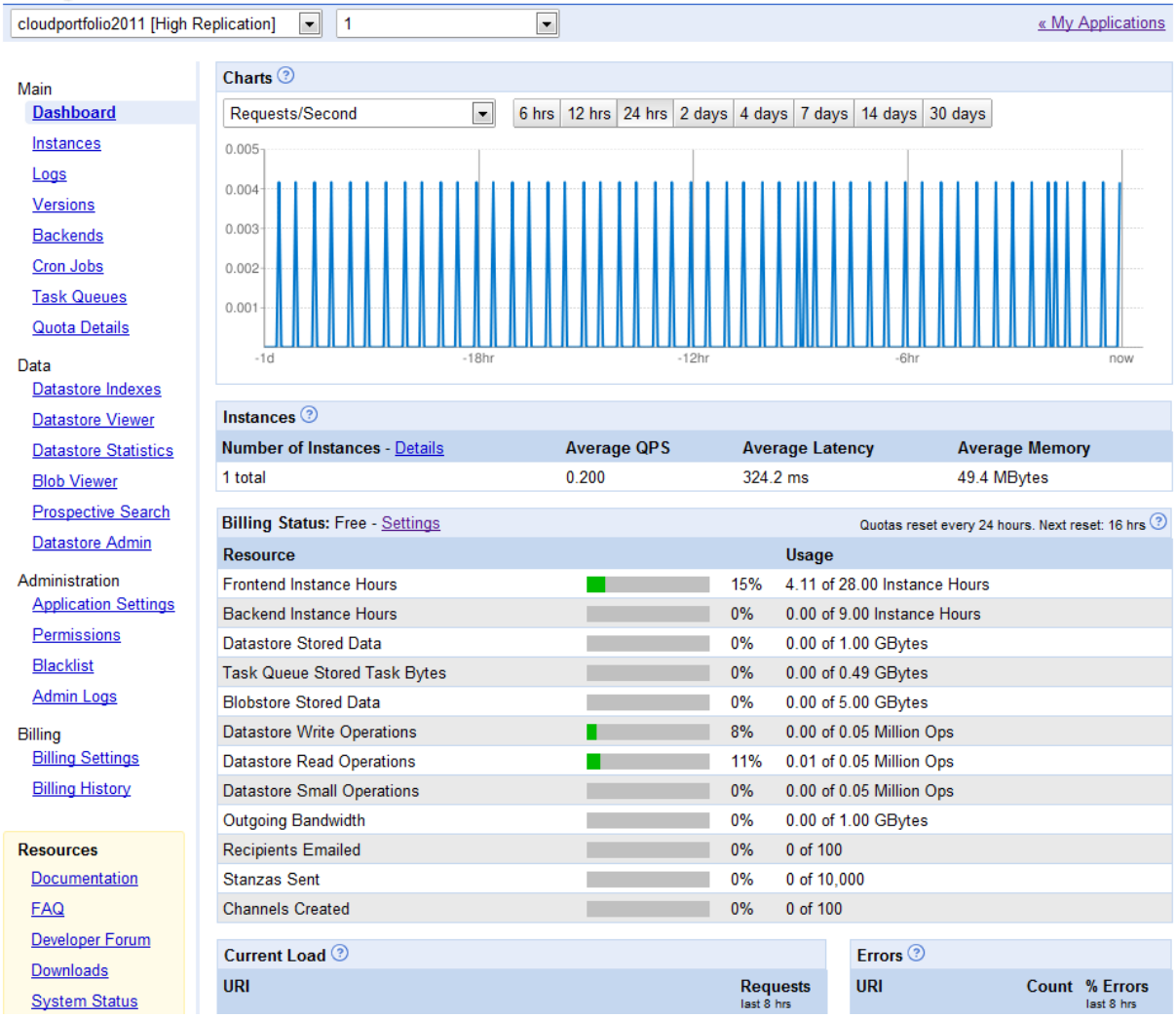
SDK za Javo, Python in Go vsebujejo spletni strežnik, ki na lokalnem računalniku omogoča razvoj in testiranje aplikacije, simulira podatkovno bazo Datastore in vse storitve App Engine. Samodejno se lahko na podlagi poizvedb, ki jih aplikacija izvede tekom testiranja, generira tudi datoteka, ki vsebuje definirane indekse za podatkovno bazo. Spletni strežnik simulira tudi peskovnik, vključno s preprečevanjem dostopov do sistemskih virov, ki v dejanskem App Engine izvajalnem okolju niso dovoljeni.

Vsak SDK vsebuje tudi orodje za nalaganje aplikacije na App Engine. Po tem, ko kreiramo aplikacijsko kodo, statične datoteke in nastavitvene datoteke, jih lahko s tem orodjem naložimo na App Engine. Za uporabo orodja se je potrebno identificirati z e-poštnim naslovom in geslom našega Google računa.

Vsako novo izdajo naše aplikacije lahko na App Engine naložimo kot novo verzijo. Medtem ko testiramo novo verzijo, lahko uporabniki še vedno naprej uporabljajo staro verzijo, dokler je ne deaktiviramo.

S pomočjo administracijske konzole (slika 3.7) lahko upravljamo z aplikacijami, ki so naložene na platformi.

Implementacija večnajemniškega modela
Za implementacijo večnajemniškega modela App Engine uporablja pristop 1: deljenje vmesnega sloja programske opreme z eno samo instanco aplikacije.



Slika 3.7. Administracijska konzola za Google App Engine

3.5.4 Model zaračunavanja storitev

Google uporablja model zaračunavanja na podlagi porabe virov (*Pay-as-you-go*). Vsak vir ima določeno brezplačno kvoto, ki se dnevno obnavlja. Če je za vir vključeno zaračunavanje, potem se količina porabljenega vira nad to brezplačno kvoto zaračuna po ceniku. Če zaračunavanje ni vključeno, potem aplikacija, ko porabi brezplačno kvoto vira, tega ne more več uporabljati vse dokler se ne obnovi.

3.6 Primerjava rešitev PaaS

Trenutno se na trgu PaaS odvijajo nenehne spremembe in ponudniki rešitev PaaS so še v fazi raziskovanja uporabnikovih zahtev. V diplomski nalogi smo obravnavali štiri ponudnike in njihove platforme: Microsoft Windows Azure, Salesforce.com Force.com, AWS Elastic Beanstalk in Google App Engine.

Primerjava med njimi pokaže, da vsak zase ponujajo precej specifične rešitve PaaS, ki se med sabo razlikujejo po implementaciji večnajemniškega modela, funkcionalnostih, naboru programskih orodij, modelu zaračunavanja storitev in ceni. Vsaka rešitev je namenjena določenemu krogu uporabnikov.

Windows Azure je platforma, ki cilja na obstoječe razvijalce Windows aplikacij, ki uporabljajo predvsem Microsoftove tehnologije.

Salesforce.com Force.com je namenjena predvsem nadgradnji obstoječih Salesforce.com aplikacij, ki jih uporabljajo velike organizacije, čeprav se v zadnjem času priazdeva tudi za širitev izven tega okvira.

AWS Elastic Beanstalk je najmlajši predstavnik izmed štirih in omogoča razvoj spletnih aplikacij, ki temeljijo na Javi.

Google App Engine je privlačna rešitev za širok krog razvijalcev, saj (z določenimi omejitvami) ponuja tudi časovno neomejeno brezplačno platformo za razvoj in izvajanje aplikacij, njen fokus pa so predvsem visoko skalabilne spletne aplikacije.

Zgoščen pregled in primerjavo med vsemi štirimi obravnavanimi rešitvami PaaS podaja spodnja preglednica 3.1:

Preglednica 3.1. Primerjava funkcionalnosti obravnavanih rešitev PaaS

PaaS rešitev	Windows Azure	Salesforce Force.com	AWS Elastic Beanstalk	Google App Engine
Vstop na trg PaaS	Februar 2010	September 2007	Januar 2011	April 2008
Tip platforme PaaS	Samostojna platforma	Platforma za nadgrajevanje obstoječih aplikacij	Samostojna platforma	Samostojna platforma
Podprti programski jeziki	C#, Java, Ruby, itd.	Lasten programski jezik Apex, Java (VMForce)	Java	Python, Java, lasten programski jezik Go
Implementacija večnajemniškega modela	Pristopi 1, 2, 3	Pristop 1	Pristop 4	Pristop 1
Razvojno okolje	Visual Studio, poljubno	Spletno okolje, Force.com IDE, Visual Force (načrtovanje UI)	Poljubno	Poljubno
Tip razvojnega okolja	Lokalno	Lokalno in spletno	Lokalno	Lokalno
Podatkovna baza	SQL Azure (relacijska PB), Windows Azure Storage (nerelacijska PB)	Nerelacijska PB	Amazon RDS (relacijska PB), Simple DB (nerelacijska PB)	Google BigTable (nerelacijska PB), Google Cloud SQL (relacijska PB, v razvoju)
Infrastruktura	Lastna	Lastna	Lastna	Lastna
Storitve	Windows Azure, Windows Azure AppFabric, Microsoft SQL Azure, Windows Azure Marketplace, DataMarket	Apex Web Services, Force.com Web Services, Force.com Toolkit for Azure, Force.com for Amazon Web Services, Force.com for Google App Engine, Callouts and Mashups, Force.com Code Share	Amazon EC2, Amazon S3, Amazon SimpleDB, Amazon SQS, Amazon RDS, Amazon Elastic MapReduce	Google Cloud SQL (v razvoju), Google BigQuery, Google Cloud Storage (eksperiment, samo za Python), Prospective Search, MapReduce (eksperiment, samo za Python), App Identity, Blobstore, Capabilities, Channel, Images, Mail, Memcache, Multitenancy, OAuth, Task Queues, URL Fetch, Users, XMPP

Možnost povezovanja z ostalimi platformami	Da, uporaba storitve PB na Force.com (potreben je Force.com Toolkit for Azure)	Da, uporaba storitve PB Amazon S3 in storitve računanja Amazon EC2 na AWS Elastic Beanstalk (potreben je Force.com for Amazon Web Services)	Ne	Da, uporaba storitve PB na Force.com (potreben je Force.com for Google App Engine)
Model zaračunavanja storitev	Naročnina, količina porabljenih virov	Naročnina	Količina porabljenih virov	Količina porabljenih virov
Fokus	Spletne aplikacije za Windows okolje	Nadgradnja obstoječih Salesforce.com aplikacij, v zadnjem času prizadevanja tudi za razvoj aplikacij izven okvira Salesforce.com	Java spletne aplikacije	Visoko skalabilne spletne aplikacije

Gledano z vidika varnosti, vsak izmed obravnavanih ponudnikov rešitev PaaS nudi dobro podporo za razvoj varnih aplikacij. Uporabniki, torej razvijalci, s tega stališča pri odločitvi za razvoj in gostovanje svojih aplikacij na katerekoli izmed štirih rešitev PaaS ne bodo prikrajšani. Vendar pa na manjše zaupanje v ponudnika lahko vpliva odsotnost dogovora SLA, ki med drugim opredeljuje nivo zagotovljene razpoložljivosti storitev. Izmed štirih ponudnikov ga za svoje storitve opredeljujeta Microsoft Windows Azure in Google App Engine, AWS Elastic Beanstalk le za nekatere storitve, Salesforce.com Force.com pa ga ne opredeljuje.

Primerjavo obravnavanih ponudnikov rešitev PaaS s stališča varnosti podaja spodnja preglednica 3.2:

Preglednica 3.2. Primerjava obravnavanih rešitev PaaS s stališča varnosti

PaaS rešitev	Windows Azure	Salesforce Force.com	Amazon AWS Beanstalk	Google App Engine
SLA	DA	NE	Samo za nekatere storitve	DA
Avtentikacija	Uporabniško ime in geslo, Windows Live ID, Google Accounts, povezana identifikacija, certifikati, itd.	Uporabniško ime in geslo, povezana identifikacija (SAML), LDAP, itd.	Uporabniško ime in geslo, par ključev, povezana identifikacija	Uporabniško ime in geslo (Google Accounts), povezana identifikacija (OpenID), lastno razvito ogrodje za avtentikacijo
Avtorizacija	Avtorizacijska pravila	SAML, profili, avtorizacijska pravila	Varnostne skupine	Uporabniške vloge, OAuth
Integriteta	Avtentikacija, enkripcija (SSL, različni kriptografski mehanizmi)	Avtentikacija, enkripcija (SSL)	Avtentikacija, enkripcija (SSL)	Avtentikacija, enkripcija (SSL)
Zaupnost	Varnostna politika, enkripcija, avtentikacija	Varnostna politika, enkripcija, avtentikacija	Varnostna politika, enkripcija, avtentikacija	Varnostna politika, enkripcija, avtentikacija
Razpoložljivost	Je podana s SLA, različna glede na posamezne storitve (večinoma 99.9%)	Ponudnik navaja 99.9% razpoložljivost, ni SLA	Je podana s SLA, vendar ne za vse storitve, sicer različna glede na posamezne storitve (večinoma 99.9%)	Je podana s SLA (99.95%)

Poglavje 4

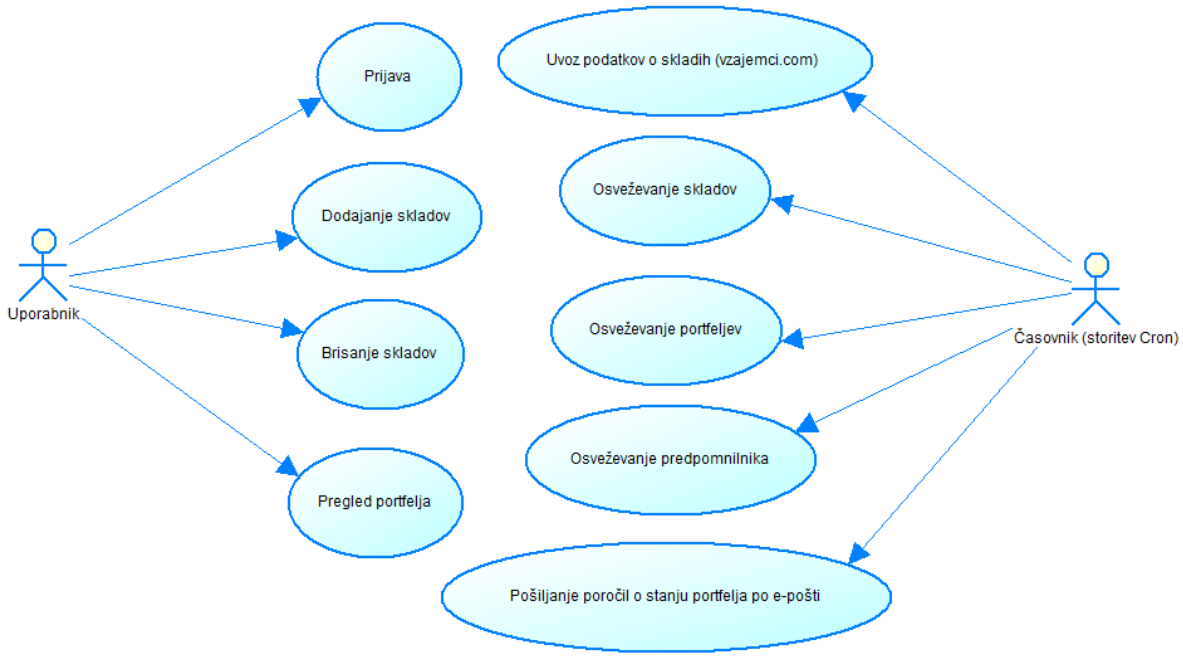
Razvoj spletne aplikacije in postavitev na platformi Google App Engine

4.1 Uvod

V praktičnem delu bomo razvili spletno aplikacijo *Portfelj v oblaku*, s pomočjo katere bomo preizkusili značilnosti izbrane rešitve PaaS. Od štirih ponudnikov platform v oblaku, ki smo jih obravnavali v prejšnjem poglavju, bomo izbrali Google in njegovo platformo App Engine. Razlog za izbiro je podpora programskemu jeziku Java, s katerim imamo že izkušnje in cena, saj je platforma za naše potrebe na voljo brezplačno.

4.2 Opis aplikacije

Spletna aplikacija *Portfelj v oblaku* bo prijavljenemu uporabniku omogočala dodajanje skladov v portfelj, brisanje skladov iz portfelja ter spremljanje stanja portfelja. Aplikacija bo samodejno poskrbela za dnevno osveževanje vseh skladov v bazi z uvažanjem podatkov s spletne strani vzajemci.com ter za posodabljanje portfeljev uporabnikov. Vsak dan bo uporabnik dobil tudi e-poštno sporočilo s poročilom o stanju svojega portfelja. Slika 4.1 prikazuje diagram primerov uporabe aplikacije *Portfelj v oblaku*:



Slika 4.1. Diagram primerov uporabe aplikacije Portfelj v oblaku

4.3 Opis uporabljene tehnologije

4.3.1 Programski jezik

Spletna aplikacija bo razvita v programskem jeziku Java. Zaenkrat Google App Engine omogoča razvoj javanskih aplikacij za verziji Java 5 in Java 6. Za razvoj in nalaganje aplikacije na App Engine imamo na voljo nabor orodij App Engine Java SDK. Ta vsebuje spletni aplikacijski strežnik, ki ga poganjamo lokalno za testiranje aplikacije. Strežnik lahko simulira vse storitve App Engine, vključno z lokalno podatkovno bazo Datastore.

4.3.2 Izvajalno okolje

App Engine izvaja javanske aplikacije s pomočjo javine navidezne naprave (*Java Virtual Machine, JVM*). Ker uporablja JVM od verzije Java 6, je priporočljivo, da se Javo 6 uporablja tudi pri razvoju in testiranju aplikacije na lokalnem spletnem strežniku. S tem zagotovimo, da se aplikacija na App Engine ne bo obnašala bistveno drugače kot na lokalnem spletnem strežniku.

4.3.3 Razvojno okolje

Za razvojno okolje bomo uporabili Eclipse 3.6 Helios [24], z integriranim vtičnikom Google Plugin for Eclipse, ki omogoča razvoj, testiranje in nalaganje aplikacije na App Engine.

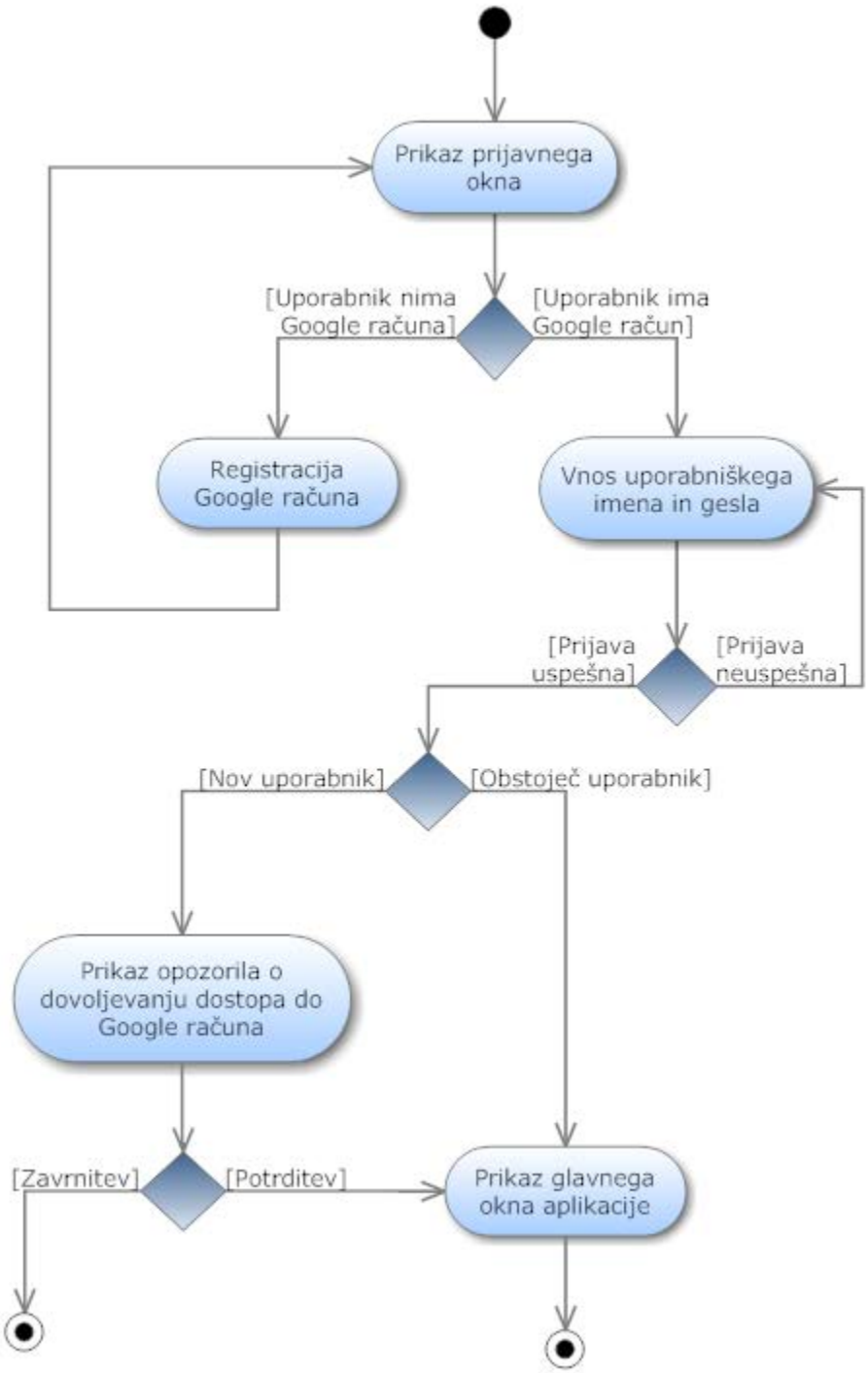
4.3.4 Poslovna logika

Za komunikacijo med spletno aplikacijo v Javi in spletnim strežnikom, App Engine uporablja servlete (*Servlets*). To so razredi, ki lahko procesirajo spletne zahteve in nanje odgovarjajo.

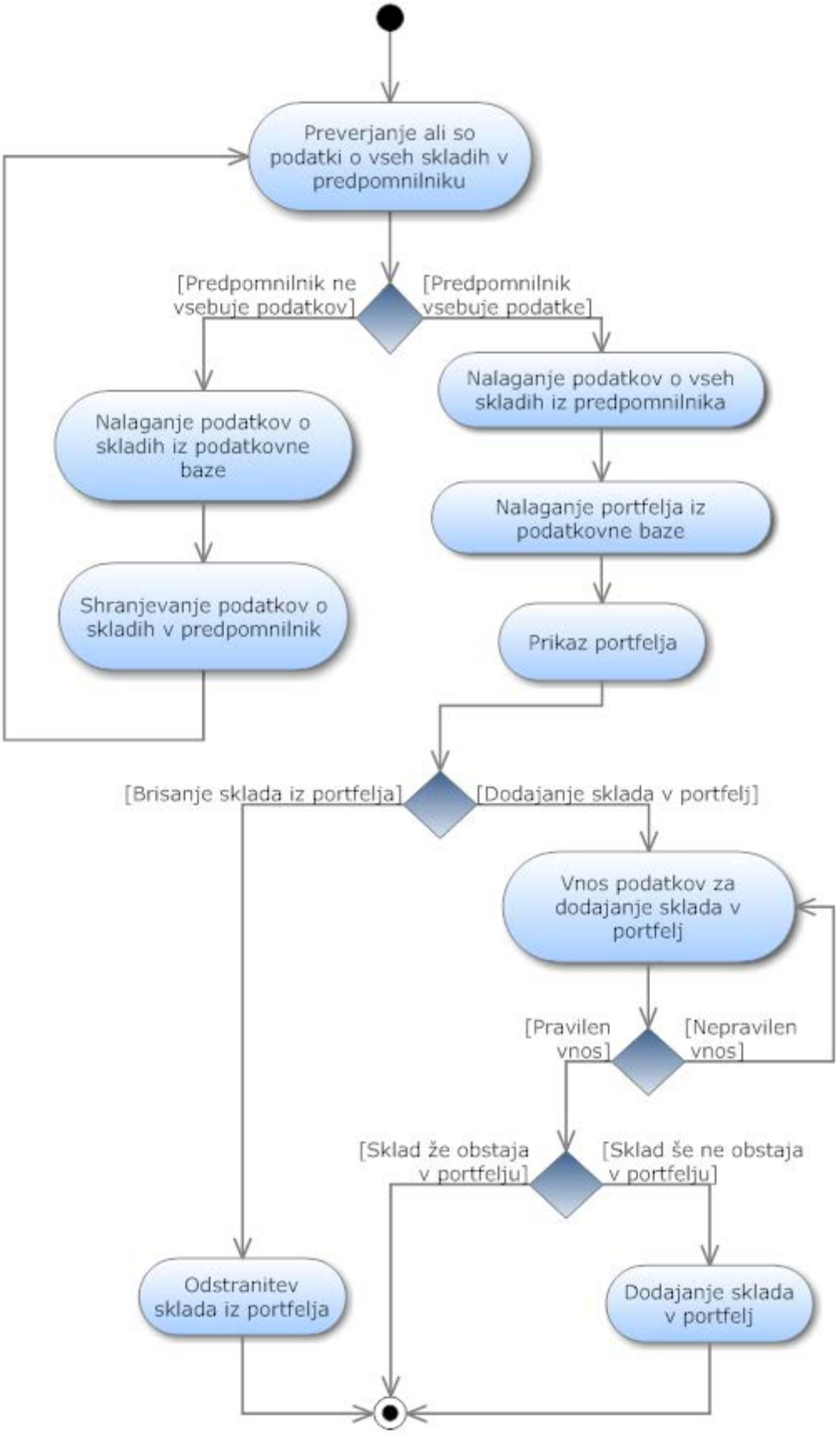
Posamezne sklope funkcionalnosti naše aplikacije smo implementirali v ločenih servletih, ki so sledeči:

- **PortfolioServlet.java**: glavni servlet, ki sprejema zahteve uporabnikov naše aplikacije in je tako zadolžen za prijavo in odjavo uporabnikov, prikazovanje portfelja ter dodajanje in brisanje skladov iz portfelja.
- **UpdateServlet.java**: skrbi za uvoz dnevno osveženih podatkov o skladih iz spletne strani vzajemci.com, posodabljanje baze podatkov o vseh skladih in posodabljanje portfeljev uporabnikov. Servlet ne sprejema zahtev uporabnikov, ampak ga v točno določenem času pokliče App Engine v okviru storitve Cron, ki služi kot časovnik.
- **CacheServlet.java**: skrbi za periodično osveževanje predpomnilnika, v katerem so shranjeni podatki o vseh skladih, ki se pogosto uporabljajo. Servlet pokliče App Engine v okviru storitve Cron.
- **SendEmailServlet.java**: skrbi za vsakodnevno pošiljanje poročil o stanju portfeljev uporabnikov po e-pošti. Servlet pokliče App Engine v okviru storitve Cron.

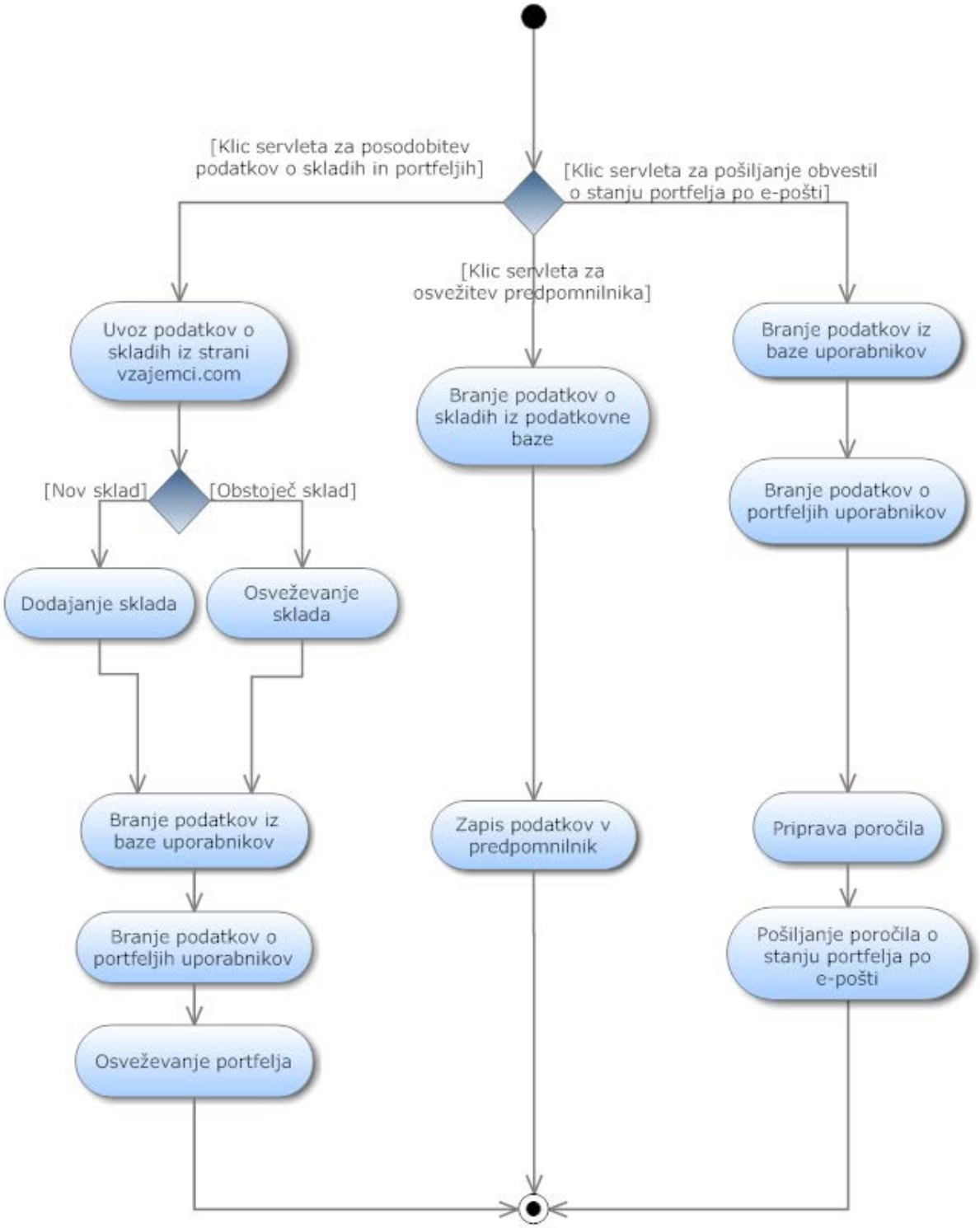
Za lažjo ponazoritev delovanja aplikacije sledijo diagram aktivnosti postopka prijave v aplikacijo (slika 4.2), diagram aktivnosti za operacije uporabe portfelja (slika 4.3) in diagram aktivnosti za operacije, ki jih izvaja časovnik oziroma storitev Cron.



Slika 4.2. Diagram aktivnosti za postopek prijave v aplikacijo



Slika 4.3. Diagram aktivnosti za operacije uporabe portfelja



Slika 4.4. Diagram aktivnosti za operacije časovnika (storitev Cron)

4.3.5 Nastavitvene datoteke

Ko spletni strežnik dobi zahtevo, na podlagi posebne nastavitvene datoteke ugotovi, kateri servlet mora klicati, da bo postregel zahtevo. Ta datoteka se imenuje *web.xml* in predstavlja t.i. postavitveni deskriptor spletne aplikacije. Izsek kode iz datoteke *web.xml* prikazuje spodnja slika 4.5:

```
	<servlet>
		<servlet-name>Portfolio</servlet-name>
		<servlet-
class>portfolio.PortfolioServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>Portfolio</servlet-name>
		<url-pattern>/portfolio</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>portfolio</welcome-file>
	</welcome-file-list>

	<filter>
		<filter-name>NamespaceFilter</filter-name>
		<filter-
class>multitenancy.NamespaceFilter</filter-class>
	</filter>
	<filter-mapping>
		<filter-name>NamespaceFilter</filter-name>
		<url-pattern>/*</url-pattern>
	</filter-mapping>

	<security-constraint>
		<web-resource-collection>
		<web-resource-name>Cron</web-resource-name>
		<description>Restricting access to cron servlets
to only admins</description>
		<url-pattern>/cron/*</url-pattern>
		</web-resource-collection>
		<auth-constraint>
			<role-name>admin</role-name>
		</auth-constraint>
	</security-constraint>
```

Slika 4.5. Datoteka web.xml

App Engine potrebuje še eno nastavitveno datoteko, na podlagi katere lahko postavi in izvede aplikacijo. Imenuje se *appengine-web.xml*. Ta datoteka med drugim vsebuje registrirano identifikacijsko številko naše aplikacije, številko verzije aplikacije, nastavitve seje itd. Datoteka *appengine-web.xml* je specifična za App Engine in ni del standarda Java Servlets. Izsek kode iz datoteke *appengine-web.xml* prikazuje slika 4.6:

```
<application>cloudportfolio2011</application>
<version>1</version>

<!-- Configure java.util.logging -->
<system-properties>
	<property name="java.util.logging.config.file"
value="WEB-INF/logging.properties"/>
</system-properties>
<sessions-enabled>true</sessions-enabled>
```

Slika 4.6. Datoteka appengine-web.xml

Vse datoteke, ki jih aplikacija potrebuje, vključno s prevedenimi razredi, knjižnicami, statičnimi datotekami (kot so CSS predloge in slike) ter nastavitvenimi datotekami, so organizirane v strukturi WAR (*Web Application Archive*).

4.3.6 Uporabniški vmesnik

Za implementacijo uporabniškega vmesnika bomo uporabili JavaServer Pages (JSP). Gre za javansko tehnologijo za izdelavo dinamičnih spletnih strani, ki je del standarda Java Servlets. JSP strani bodo služile za prikaz rezultatov, ki jih bodo posredovali servleti.

Stran JSP je v osnovi HTML dokument s posebnimi odseki, imenovanimi JSP elementi, ki so od ostale vsebine ločeni s posebnimi oznakami in vsebujejo javansko kodo. App Engine samodejno prevede JSP datoteke iz strukture WAR in jih preslika v URL naslove. Izsek kode iz jsp datoteke *portfolio.jsp* prikazuje slika 4.7:

```
<form action="/portfolio" method="post">
<tr id="rowPortfolioFond">
<td><%=fondData.getFondName() %></td>
<td><%=sdf.format(fondData.getFondDateOfEntry()) %></td>
<td><%=fondData.getFondValueCurrent() %></td>
<%
	String color = "white";
	if(fondData.getFondChange() < 0){
		color = "red";
	}
	else{
		color = "green";
	}
%>
<td style="color: <%=color %>; "><b><%=fondData.getFondChange() %></b></td>
<td><%=sdf.format(fondData.getFondDateOfValue()) %></td>
<td><%=fondData.getFondNumOfPoints() %></td>
<td><%=fondData.getFondValueStart() %></td>
```

Slika 4.7. Datoteka portfolio.jsp

V kombinaciji z JSP stranmi bomo uporabili še jQuery UI [8], ki je knjižnica JavaScript funkcij za oblikovanje uporabniškega vmesnika.

4.3.7 Podatkovna baza

Podatkovni model
Za potrebe naše aplikacije potrebujemo naslednje entitetne tipe:

- **FondDB:** vsebuje podatke o vseh skladih.
- **PortfolioFond:** vsebuje podatke o skladih, ki so v uporabnikovem portfelju.
- **Users:** vsebuje podatke o uporabnikih aplikacije.

Podrobnosti podatkovnega modela podaja spodnja preglednica 4.1:

Preglednica 4.1. Podatkovni model aplikacije Portfelj v oblaku

FondDB		
Lastnost	**Tip vrednosti**	**Opis**
fondDateOfRefresh	Date/Time	Datum zadnje osvežitve
fondDateOfValue	Date/Time	Datum vrednosti sklada
fondValueCurrent	Float	Trenutna vrednost sklada
fondChange	Float	Dnevna sprememba sklada
fondName	String	Ime sklada
date	Date/Time	Datum vnosa v bazo

PortfolioFond		
Lastnost	**Tip vrednosti**	**Opis**
fondDateOfRefresh	Date/Time	Datum zadnje osvežitve
fondDateOfEntry	Date/Time	Datum vstopa v sklad
fondDateOfValue	Date/Time	Datum vrednosti sklada
fondValueCurrent	Float	Trenutna vrednost sklada
fondNumOfPoints	Float	Število točk
fondValueStart	Float	Začetna vrednost premoženja v skladu
fondChangeSum	Float	Sprememba med začetno in trenutno vrednostjo premoženja v skladu
fondValueSum	Float	Trenutna vrednost premoženja v skladu
fondChange	Float	Dnevna sprememba vrednosti sklada
fondName	String	Ime sklada
date	Date/Time	Datum vnosa v bazo

Users		
Lastnost	**Tip vrednosti**	**Opis**
User	User	Uporabnik
dateAdded	Date/Time	Datum vnosa v bazo

Indeksi

Datastore uporablja indekse za vsako poizvedbo, ki jo aplikacija izvede. Ti indeksi se posodobijo vsakič, ko se entiteta spremeni, kar omogoča hitro izvedbo poizvedbe. Za ta namen mora Datastore vnaprej vedeti, katere poizvedbe lahko aplikacija vse izvaja. Indeksi za poizvedbo so podani v nastavitveni datoteki *datastore-indexes.xml*.

Namesto da ročno definiramo indekse, lahko za to poskrbi kar lokalni spletni strežnik, na katerem aplikacijo testiramo. Če aplikacija izvede poizvedbo, ki nima ustreznega indeksa, ga strežnik samodejno kreira in doda njegovo definicijo v datoteko *datastore-indexes-auto.xml*. Če želimo poleg ročno definiranih indeksov uporabljati tudi samodejno generiranje indeksov, moramo v nastavitveni datoteki *datastore-indexes.xml* nastaviti atribut autoGenerate na "true".

Za potrebe naše aplikacije smo omogočili samodejno generiranje indeksov, ročno pa smo kreirali sledeče indekse:

- FondDB: dva indeksa po imenu sklada (fondName) in datumu vnosa v bazo (date)
- PortfolioFond: dva indeksa po imenu sklada (fondName) in datumu vnosa v bazo (date)

Izsek kode iz datoteke *datastore-indexes.xml*, ki vsebuje definicijo indeksov, prikazuje slika 4.8:

```
<datastore-indexes
        autoGenerate="true">
       <datastore-index kind="FondDB" ancestor="false">
              <property name="fondName" direction="asc" />
              <property name="date" direction="desc" />
        </datastore-index>

       <datastore-index kind="PortfolioFond" ancestor="false">
              <property name="fondName" direction="asc" />
              <property name="date" direction="desc" />
       </datastore-index>
/datastore-indexes>
```

Slika 4.8. Datoteka datastore-indexes.xml

4.4 Opis funkcionalnosti aplikacije

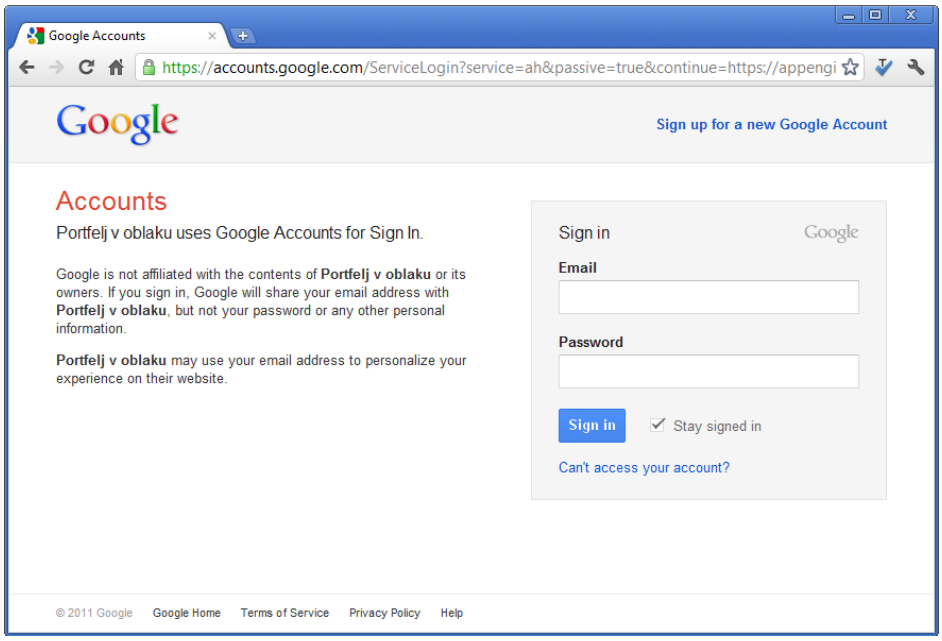
Avtentikacija uporabnikov aplikacije

Za naše potrebe bomo omogočili avtentikacijo uporabnikov na podlagi njihovih Google računov. Za ta namen imamo na voljo Users API, na podlagi katerega lahko ugotovimo, ali je uporabnik prijavljen s svojim računom ali ne. Izsek kode za avtentikacijo uporabnikov prikazuje slika 4.9:

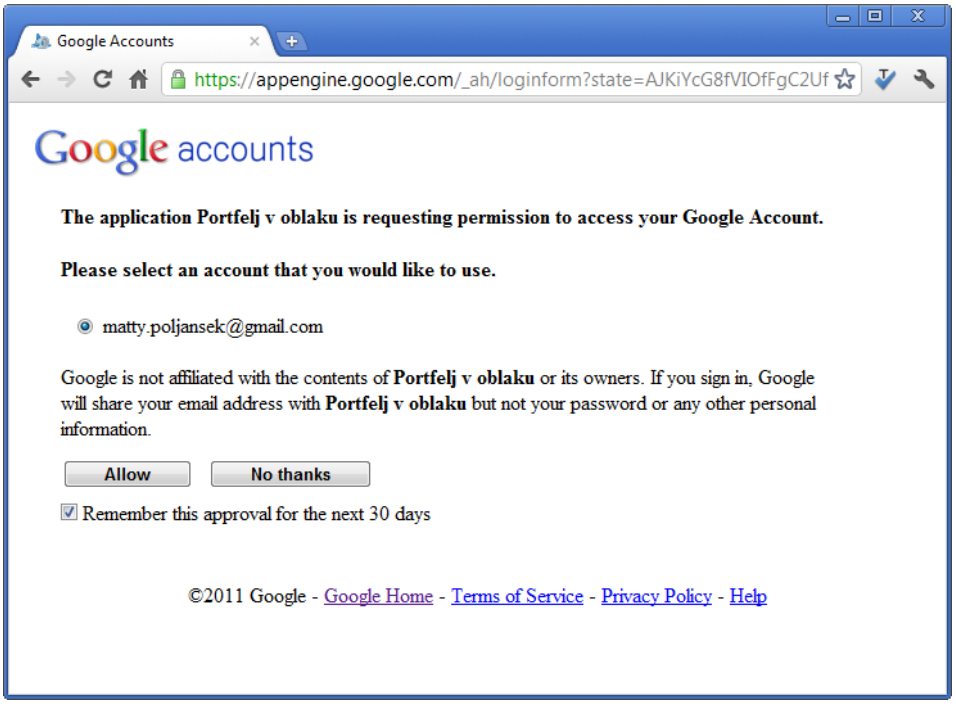
```
<% //get current user
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
if (user != null) {
        String destination = "/portfolio";
        response.sendRedirect(response.encodeRedirectURL(destination));
        return;
}
else { %>
        <table class="tableMain" cellpadding="2px" cellspacing="1px">
        <tr>
        <td><h2>Dobrodošli v aplikaciji <b>Portfelj v oblaku</b>!</h2></td>
        </tr>
        <tr>
        <td><p>Za uporabo aplikacije se morate prijaviti.</p></td>
        </tr>
        <tr>
        <td><p>Prijavite se lahko <a
href="<%=userService.createLoginURL(request.getRequestURI())%>">tukaj</a>.</p></td>
        </tr>
        </table>
```

Slika 4.9. Avtentikacija uporabnikov

Če uporabnik še ni prijavljen, ga storitev preusmeri na prijavno stran s prilagojeno vsebino za našo aplikacijo, ali pa mu omogoči, da kreira nov Google račun (slika 4.10). Novemu uporabniku se prikaže še obvestilo, v katerem se zahteva potrditev, da aplikacija lahko dostopa do njegovega Google računa (slika 4.11). Po uspešni prijavi ali kreiranju računa, se uporabnika preusmeri na glavno stran aplikacije.



Slika 4.10. Prijava v aplikacijo Portfelj v oblaku



Slika 4.11. Opozorilno okno za novo prijavljene uporabnike

Vsakodnevno avtomatsko izvajanje opravil

App Engine nudi storitev Cron, s pomočjo katerega lahko nastavimo, da se določeni servleti ob izbranem časovnem intervalu samodejno pokličejo.

V naši aplikaciji bomo storitev Cron uporabili za vsakodnevno osveževanje podatkov o skladih in uporabnikovega portfelja (*UpdateServlet.java*), polnjenje predpomnilnika (*CacheServlet.java*) ter pošiljanje poročila o stanju uporabnikovega portfelja po e-pošti (*SendEmailServlet.java*). Za ta namen je potrebno v datoteki *cron.xml* podati določene nastavitve. Izsek kode iz datoteke *cron.xml* prikazuje slika 4.12:

```
<cronentries>
  <cron>
    <url>/update</url>
    <description>Update fond database and user portfolios every day at 15:00
    </description>
    <schedule>every day 15:00</schedule>
    <timezone>Europe/Ljubljana</timezone>
  </cron>
  <cron>
    <url>/cache</url>
    <description>Repopulate the cache every 30 minutes</description>
    <schedule>every 30 minutes</schedule>
    <timezone>Europe/Ljubljana</timezone>
  </cron>
  <cron>
    <url>/email</url>
    <description>Send email to users every day during the week at 08:00
    </description>
    <schedule>every monday,tuesday,wednesday,thursday,friday 08:00</schedule>
    <timezone>Europe/Ljubljana</timezone>
  </cron>
</cronentries>
```

Slika 4.12. Datoteka cron.xml

Omejevanje dostopa do resursov

Poleg omejevanja dostopa do celotne aplikacije samo avtenticiranim uporabnikom, lahko na podlagi uporabnikovega Google računa dodelimo omejitve dostopa tudi za posamezne URL naslove s pomočjo posebnega deskriptorja v *web.xml* datoteki. Google App Engine to omogoča samo za dve vlogi – navadni uporabniki in administratorji - lastnih varnostnih vlog ne podpira.

V našem primeru bomo navadnim uporabnikom aplikacije onemogočili dostop do servletov za posodabljanje podatkov o skladih in portfeljih (*UpdateServlet.java*), nalaganje v predpomnilnik (*CacheServlet.java*) in pošiljanje obvestil po e-pošti (*SendEmailServlet.java*). Do teh bo tako lahko dostopal samo administrator. Izsek kode za omejevanje dostopa do servletov iz datoteke *web.xml* prikazuje slika 4.13:

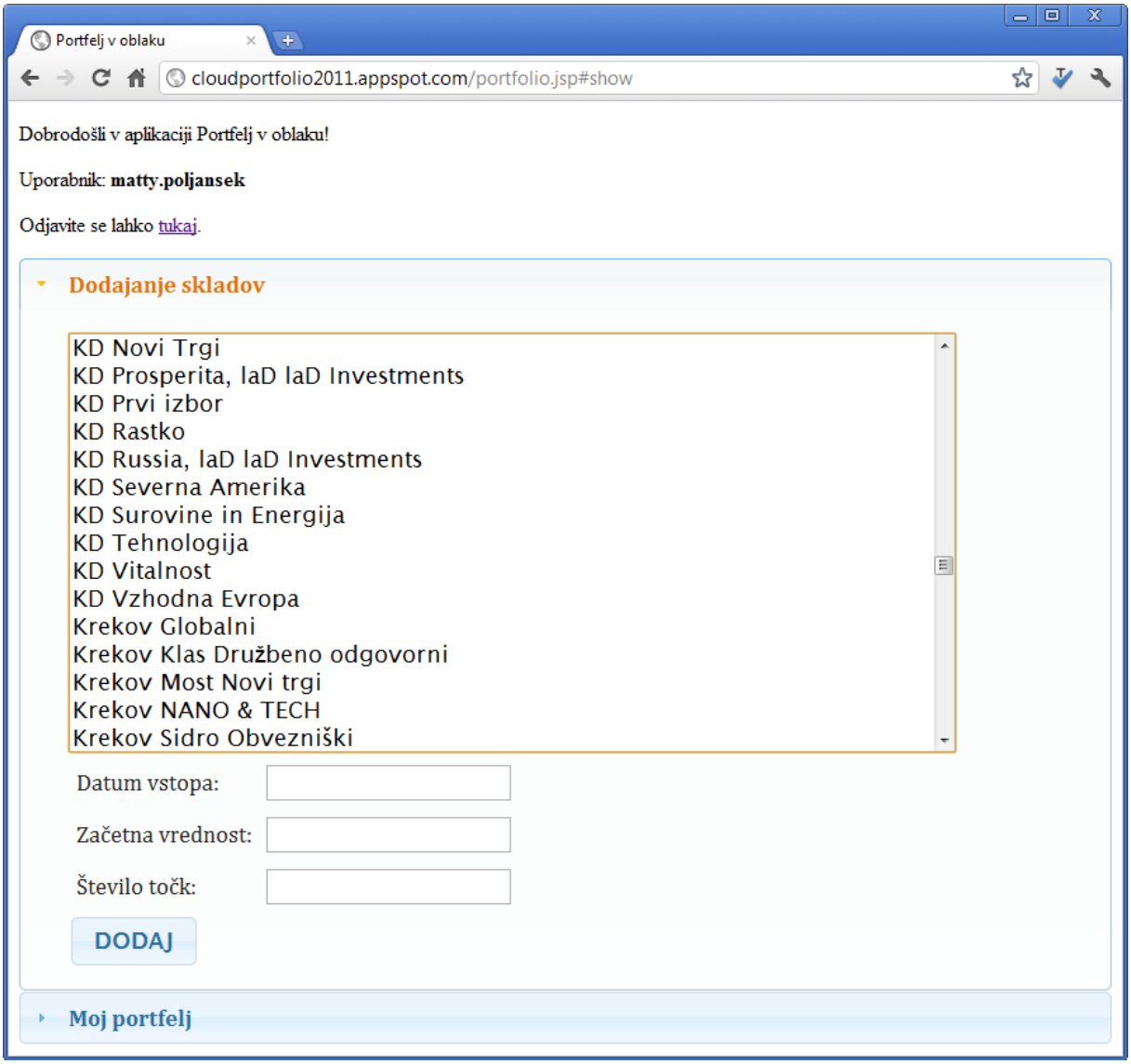
```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Cron</web-resource-name>
    <description>Restricting access to cron servlets only to admins
    </description>
    <url-pattern>/cron/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

Slika 4.13. Omejevanje dostopa do servletov

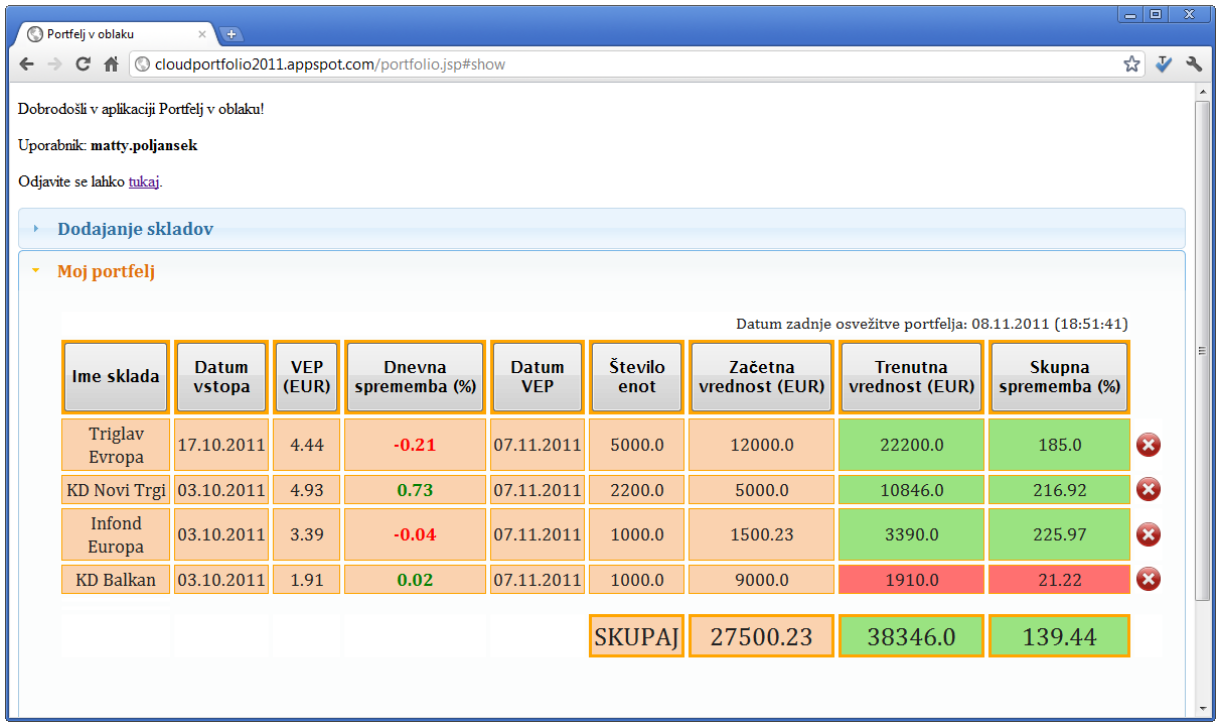
Uporaba portfelja

Uporabnik, ki se uspešno prijavi s svojim Google računom, lahko dodaja sklade v portfelj (slika 4.14), tako da s seznama skladov izbere sklad, vpiše podatke kot so datum vstopa v sklad, število enot in začetno vrednost premoženja v skladu. V primeru neizbranega sklada ali neustreznega vnosa podatkov, aplikacija uporabnika ustrezno opozori.

Uporabnikov portfelj (slika 4.15) se dnevno osvežuje na podlagi podatkov s spletne strani vzajemci.com, uporabnik pa lahko sklade s portfelja tudi briše. Vse glavne funkcionalnosti aplikacije izvaja servlet *PortfolioServlet.java*.



Slika 4.14. Dodajanje skladov v aplikaciji Portfelj v oblaku



Slika 4.15. Uporabnikov portfelj v aplikaciji Portfelj v oblaku

Posodabljanje podatkov o skladih in portfeljih

Za posodabljanje podatkov o skladih in portfeljih skrbi servlet *UpdateServlet.java*. Ta se pokliče ob določenih časovnih intervalih, ki so podani v datoteki *cron.xml*. Najprej se iz spletne strani vzajemci.com naloži njena HTML datoteka, ki vsebuje najbolj sveže podatke o skladih. Iz te HTML datoteke se s pomočjo knjižnice *jsoup* [6] izluščijo podatki o skladih, na podlagi katerih lahko nato osvežimo najprej našo bazo skladov, nato pa še vse sklade, ki so v portfeljih uporabnikov.

Nalaganje in osveževanje podatkov predpomnilnika

Podatke o vseh skladih, ki se vedno uporabljajo, shranjujemo v predpomnilnik, kjer jih periodično osvežujemo. Za to skrbi servlet *CacheServlet.java*. Ta prebere podatke o skladih iz baze in jih s pomočjo App Engine storitve MemCache naloži v predpomnilnik. Servlet se kliče v določenih časovnih intervalih, ki so podani v datoteki *cron.xml*,. Podatki v predpomnilniku zaradi določenih razlogov kot so prevelika zasedenost predpomnilnika, niso vedno na voljo. V takih izjemnih primerih je potrebno zagotoviti nemoteno delovanje aplikacije. V naši aplikaciji je za to poskrbljeno v okviru servleta *PortfolioServlet.java*, ki v primeru, da podatki v predpomnilniku niso na voljo, le-te prebere iz podatkovne baze.

Pošiljanje e-poštnih obvestil

Uporabniku se vsak dan ob določenem času, ki je naveden v datoteki *cron.xml*, pošlje obvestilo o stanju njegovega portfelja. Za to poskrbi servlet *SendEmailServlet.java*. Ta za vsakega uporabnika iz podatkovne baze prebere podatke o njegovem portfelju in oblikuje poročilo, ki ga pošlje po e-pošti. To naredi s pomočjo storitve Mail (*JavaMail API*).

4.5 Implementacija večnajemniškega modela

Večnajemniški model lahko zagotovimo s pomočjo App Engine storitve Multitenancy, ki uporablja imenske prostore (*Namespaces*). Za ta namen imamo na razpolago vmesnik Namespaces API.

V naši aplikaciji bo imel vsak uporabnik oziroma najemnik svoj imenski prostor, s čimer bomo zagotovili, da bo lahko dostopal samo do svojih podatkov. Uporabnikov imenski prostor bo predstavljen z njegovo identifikacijsko številko, ki jo lahko dobimo s pomočjo storitve Users in metode getUserId(). Ta nam za prijavljenega uporabnika pridobi njegovo unikatno in nespremenljivo identifikacijsko številko.

Za nastavitev imenskih prostorov se v Javi uporablja servlet *NamespaceFilter.java*, ki implementira vmesnik Filter. Servlet bo v okviru metode doFilter poskrbel, da se imenski prostor nastavi na uporabnikovo identifikacijsko številko. Izsek kode iz servleta *NamespaceFilter.java* za nastavitev imenskih prostorov prikazuje slika 4.16:

```
public class NamespaceFilter implements javax.servlet.Filter {

  @Override
  public void doFilter(ServletRequest req, ServletResponse res, FilterChain
chain) throws IOException, ServletException {

   // set() must be called only when the current namespace is not already set.
   if (NamespaceManager.get() == null) {
      UserService userService = UserServiceFactory.getUserService();
      User user = userService.getCurrentUser();

      if(user != null){

        // set current user's unique ID as namespace
        NamespaceManager.set(user.getUserId());
      }
    }

    chain.doFilter(req, res);
  }
}
```

Slika 4.16. Servlet za nastavitev imenskih prostorov

Pot do servleta, ki nastavlja imenski prostor, moramo nato podati še v nastavitveni datoteki *web.xml*. S tem smo dosegli, da bo naša aplikacija kot privzeti imenski prostor uporabljala imenski prostor na podlagi uporabnikove identifikacijske številke. Izsek kode za nastavitev poti do servleta za nastavljanje imenskih prostorov iz datoteke *web.xml* prikazuje slika 4.17:

```
<filter>
        <filter-name>NamespaceFilter</filter-name>
        <filter-class>multitenancy.NamespaceFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>NamespaceFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Slika 4.17. Pot do servleta za nastavitev imenskih prostorov

Storitev podatkovne baze Datastore uporablja privzeti imenski prostor, v našem primeru torej uporabnikovo identifikacijsko številko. Vse operacije nad podatkovno bazo tako uporabljajo podatke, ki pripadajo določenemu uporabniku s to identifikacijsko številko.

V nekaterih primerih pa želimo dostopati do podatkov, ki so v drugem imenskem prostoru. Za ta namen, moramo drug imenski prostor eksplicitno podati. Tak primer so podatki o vseh skladih, ki morajo biti na voljo vsem uporabnikom aplikacije. Za njih smo določili globalen imenski prostor imenovan "global", zato moramo za dostop do teh podatkov imenski prostor začasno eksplicitno nastaviti na "global". Primer kode, ki poskrbi za eksplicitno nastavljanje imenskega prostora, prikazuje slika 4.18:

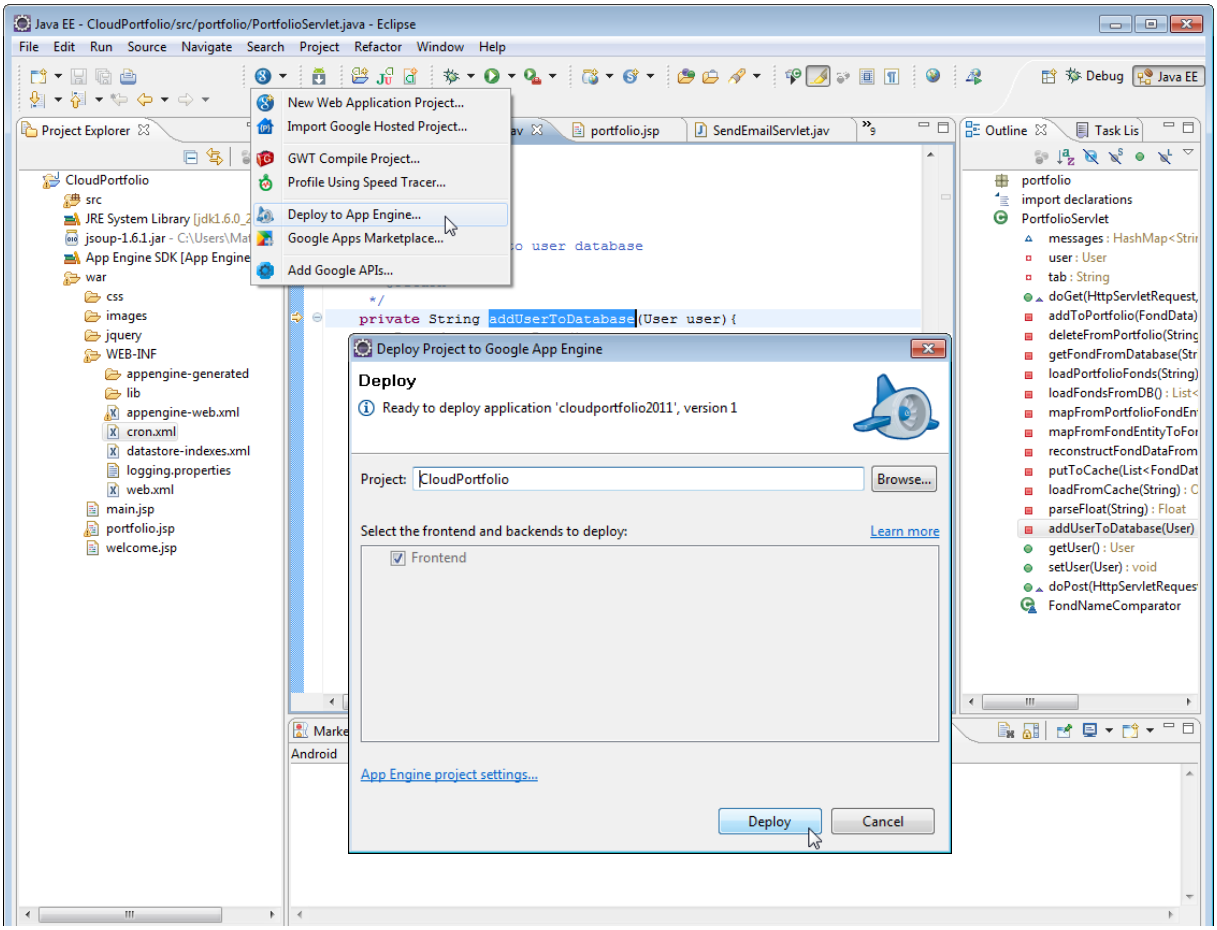
```
String oldNamespace = NamespaceManager.get();
NamespaceManager.set("global");

DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

try{
        Query query = new Query("FondDB").addSort("date",
                Query.SortDirection.DESCENDING);
        List<Entity> listOfFondEntites =
                datastore.prepare(query).asList(FetchOptions.Builder.withDefaults());

        for(Entity fondEntity : listOfFondEntites){
                FondData fondData = mapFromFondEntityToFondData(fondEntity);
                listOfFonds.add(fondData);
        }
} finally{
        NamespaceManager.set(oldNamespace);
}
```

Slika 4.18. Eksplicitno nastavljanje imenskega prostora

Globalen imenski prostor mora na podoben način uporabljati tudi predpomnilnik, kadar se ga uporablja za dostop in shranjevanje podatkov o vseh skladih.

4.6 Postavitev aplikacije

Aplikacijo postavimo na platformo App Engine zelo enostavno s pomočjo vtičnika Google Plugin for Eclipse kot prikazuje slika 4.19:



Slika 4.19. Postavitev aplikacije na Google App Engine

Ko je aplikacija postavljena na Google App Engine, jo lahko upravljamo s pomočjo administratorske konzole. Do nje dostopamo preko spletnega brskalnika na naslovu https://appengine.google.com/

Administratorska konzola nam omogoča sledeče funkcionalnosti:

- kreiranje nove aplikacije
- dodajanje drugih oseb kot razvijalcev naše aplikacije, kar jim omogoča dostop do konzole in nalaganje novih verzij aplikacije
- pregled nad prometom v zvezi z aplikacijo
- pregled nad dnevniškimi datotekami

- pregled nad podatkovno bazo in upravljanje z indeksi
- administracijo podatkovne baze
- pregled nad instancami aplikacije
- upravljanje z vrstami opravil (zaustavljanje, brisanje, poganjanje)
- upravljanje z verzijami aplikacije
- pregled nad stanjem platforme App Engine (predvidena vzdrževalna dela, razpoložljivost posameznih storitev, izpadi, ...)

4.7 Delovanje aplikacije v času vzdrževalnih del

V času vzdrževalnih del na App Engine sta storitvi podatkovne baze in predpomnilnika nedosegljivi. App Engine omogoča detekcijo takšnih izpadov, zato da jih lahko v aplikaciji primerno obravnavamo in uporabniku prikažemo ustrezno obvestilo.

V primeru nedosegljivosti storitve podatkovne baze lahko aplikacija podatke iz podatkovne baze samo bere, ne more pa jih spreminjati ali dodajati. Izsek kode, ki detektira nedosegljivost podatkovne baze prikazuje slika 4.20:

```
try{
	datastore.put(fondEntity);
}catch(CapabilityDisabledException e){
	log.info("Datastore is currently unavailable: " + e.getLocalizedMessage());
}
```

Slika 4.20. Detektiranje nedosegljivosti podatkovne baze

Nedosegljivost storitve predpomnilnika pa pomeni, da ni možno tako branje kot tudi ne shranjevanje podatkov v predpomnilnik. Izsek kode, ki detektira nedosegljivost predpomnilnika prikazuje slika 4.21:

```
String oldNamespace = NamespaceManager.get();
NamespaceManager.set("global");

MemcacheService ms = MemcacheServiceFactory.getMemcacheService();
ms.setErrorHandler(new StrictErrorHandler());

Object o;
try {
	o = ms.get(key);
} catch (com.google.appengine.api.memcache.MemcacheServiceException e) {
	this.messages.put("memcacheUnavailable", "Memcache is unavailable: " +
		e.getLocalizedMessage());
	o = null;
} finally{
	NamespaceManager.set(oldNamespace);
}
```

Slika 4.21. Detektiranje nedosegljivosti predpomnilnika

Poglavje 5

Zaključek

Internetno omrežje je danes tako razvito in razširjeno, da ga lahko postavimo ob bok javnim storitvam kot so elektrika, voda in telefonija. Selitev računalniških storitev na splet in njihova uporaba ter zaračunavanje uporabe po zgledu javnih storitev se tako zdijo logični koraki v razvoju računalništva, katerega nosilec je po našem mnenju v tem trenutku računalništvo v oblaku.

V diplomskem delu smo podrobneje obravnavali tisti del računalništva v oblaku, ki je namenjen razvijalcem aplikacij, to je model platforme kot storitve ali PaaS. Spletne aplikacije se danes lahko soočajo z več kot milijon zahtevami uporabnikov iz celega sveta. Zagotoviti ustrezno infrastrukturo, ki bo omogočala hitro skalabilnost aplikacij, je težko predvsem pa drago. Z računalništvom v oblaku in modelom PaaS se vsa kompleksnost in stroški, ki so povezani z vzpostavljanjem in vzdrževanjem infrastrukture za izvajanje aplikacij, selijo v oblak, kar pomeni, da se zdaj razvijalci lahko bolje osredotočijo na sam razvoj aplikacije. To je privlačna stvar tako za IT oddelke v velikih organizacijah, ki s tem lahko zmanjšajo obstoječe stroške, kot tudi za manjša startup podjetja, ki se podajajo v razvijalske vode in se soočajo s finančnimi omejitvami pri zagotavljanju infrastrukture za svoje aplikacije.

PaaS pa ne prinaša prednosti le za podjetja, tudi vsak posamezen razvijalec ima zdaj za relativno nizko ceno na voljo praktično neomejene količine računskih virov in platformo za razvoj in hitro postavitev zmogljivih aplikacij, ki so na voljo končnim uporabnikom ne glede na njihovo lokacijo.

V okviru diplomske naloge smo obravnavali in primerjali tudi štiri izbrane ponudnike rešitev PaaS na trgu. Microsoft Windows Azure, Salesforce.com Force.com, AWS Elastic Beanstalk in Google App Engine. Ugotovili smo, da se razlikujejo po implementaciji večnajemniškega modela, funkcionalnostih, podpori programskim jezikom, modelu zaračunavanja storitev in ceni.

Na eni izmed platform, Google App Engine, smo razvili tudi preprost primer spletne aplikacije v Javi. Glavni izziv, s katerim smo se soočili, je bil povezan s shranjevanjem podatkov. App Engine je platforma, ki je namenjena predvsem razvoju visoko skalabilnih spletnih aplikacij in (vsaj zaenkrat) ne toliko poslovnim aplikacijam. Namesto tradicionalnih relacijskih podatkovnih baz uporablja storitev Datastore, ki temelji na lastni nerelacijski in porazdeljeni podatkovni bazi BigTable. Ta zahteva nekoliko drugačen pristop pri shranjevanju in dostopanju do podatkov, kot smo ga vajeni pri relacijskem podatkovnem modelu in poizvedovalnem jeziku SQL. Soočili smo se tudi z implementacijo večnajemniškega modela, pri čemer nismo imeli večjih težav, saj Google tako kot tudi za vse storitve, ki jih omogoča App Engine, nudi dobro napisano dokumentacijo.

Po zaključenem razvoju in postavitvi naše aplikacije na platformo, je Google spremenil svoj model zaračunavanja storitev, kar je vplivalo na delovanje naše aplikacije. Problem se je pojavil pri storitvi Datastore. Namesto merjenja časa, ki ga procesor porabi za operacije nad podatkovno bazo, se zdaj meri število operacij, ki jih za shranjevanje ali dostopanje do podatka porabi podatkovna baza. Posledično je naša aplikacija že po nekaj urah zaradi prevelikega števila porabljenih bralnih operacij presegla dnevno brezplačno kvoto. Skupaj s spremembo zaračunavanja je Google objavil tudi priročnik za optimizacijo kode, s pomočjo katerega smo lahko bistveno zmanjšali število operacij nad Datastore. Omenjen problem pravzaprav nakazuje še na eno prednost modela PaaS in zaračunavanja storitev na podlagi količine porabljenih virov – če skrbimo, da je naša koda bolj kvalitetna in optimizirana, porabimo manj virov, posledično pa so manjši tudi stroški.

V splošnem je razvoj na platformi App Engine relativno enostaven. Na voljo imamo več storitev do katerih dostopamo preko API-jev, med drugim sta koristni storitvi pošiljanja e-pošte s pomočjo Gmail in avtentikacija z Google Accounts. Po drugi strani pa je dejstvo, da gre večinoma za Googlove lastne API-je, kar posledično otežuje morebitno odločitev za prehod na druge ponudnike rešitev PaaS. Z razvojem aplikacij na App Engine smo bolj ali manj priklenjeni na podjetje Google. Še posebej to velja za shranjevanje podatkov, saj je podatkovna baza na App Engine unikatno zasnovana, zato bi bilo potrebno kodo naše aplikacije za gostovanje pri drugem ponudniku v veliki meri spremeniti.

Poleg vseh prednosti, ki jih prinašajo rešitve PaaS, ne smemo zanemariti problemov kot sta že omenjena priklenitev na ponudnika in pa varnost podatkov, predvsem tistih bolj občutljive narave. Gre za težavi, ki v največji meri vplivata na odločitev za prehod na uporabo platform v oblaku. Vendar navsezadnje se trg rešitev PaaS kljub mladosti izredno hitro razvija in tudi ti problemi bodo sčasoma premagani. Omenili smo že pojav iPaaS, ki je namenjen povezovanju aplikacij, ki gostujejo na različnih platformah ter odprti vmesnik API OCCI. Za zagotavljanje varnosti podatkov pa ima večina ponudnikov že na voljo več varnostnih mehanizmov.

5.1 Sklep

Ugotovitve kažejo, da je trg rešitev PaaS trenutno še precej nestabilen, zato je izbira med posameznimi rešitvami odvisna predvsem od tega, kakšno aplikacijo želimo razviti in v kakšnem programskem jeziku. Vsak ponudnik namreč s svojo platformo v oblaku cilja na določen krog razvijalcev in na razvoj določenih vrst aplikacij.

Poleg vrste aplikacij, ki jih bomo razvijali, smo mnenja, da je pri odločitvi za razvoj in gostovanje aplikacij na razvojnih platformah v oblaku pomembna predvsem raven zaupanja do ponudnika, da bo zagotovil ustrezen nivo varnosti in razpoložljivosti svojih storitev in ustrezno ter hitro odreagiral v primeru napak in odpovedi. Pomemben korak k pridobitvi zaupanja je vsekakor prisotnost dogovora SLA, ki ga pri nekaterih ponudnikih (še) ni.

Zaključimo lahko, da PaaS prinaša mnogo prednosti za razvijalce, zaradi katerih lahko z dobro mero verjetnosti rečemo, da bo v prihodnosti, ki se zaradi izredno hitrega razvoja tega področja niti ne zdi tako oddaljena, postal nov standard za razvoj aplikacij. Dejstvo, da so na področje rešitev PaaS poleg Salesforce.com, ki je bil prvi ponudnik platform v oblaku, posegli tudi največji igralci na področju računalništva in informatike, kot so Microsoft, Amazon in Google, pa vliva še dodatno zaupanje, da bo model PaaS postal glavni model za razvoj aplikacij. Prihodnost računalništva in razvoja aplikacij je »oblačna«.

Kazalo slik

Kazalo preglednic

Literatura in viri

[1] (2011) Amazon. "AWS Elastic Beanstalk." Dostopno na: http://aws.amazon.com/elasticbeanstalk/

[2] (2011) David Chappell. "A short introduction to cloud platforms: An enterprise-oriented view." Dostopno na: http://www.davidchappell.com/CloudPlatforms--Chappell.pdf

[3] (2011) Gartner. "Gartner Says 2011 Will Be the Year of Platform as a Service." Dostopno na: http://www.gartner.com/it/page.jsp?id=1586114

[4] (2011) Google. "Google App Engine." Dostopno na: http://code.google.com/appengine/

[5] (2011) Cloud Computing Use Case Discussion Group. "Cloud Computing Use Cases Whitepaper." Dostopno na: http://opencloudmanifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf

[6] (2011) Jonathan Hedley. "jsoup." Dostopno na: http://jsoup.org/

[7] (2011) IBM. "Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 2: Approaches for enabling multi-tenancy." Dostopno na: http://www.ibm.com/developerworks/webservices/library/ws-multitenantpart2/index.html

[8] (2011) "jQuery UI." Dostopno na: http://jqueryui.com/

[9] David S. Linthicum, *Cloud Computing and SOA Convergence in Your Enterprise. A Step-by-Step Guide*, Crawfordsville: Addison-Wesley Professional, 2010, pogl. 3.

[10] (2011) Byron Ludwig, Serena Coetzee. "A comparison of platform as a service (PaaS) clouds with a detailed reference to security and geoprocessing services." Dostopno na: http://www.isprs.org/proceedings/XXXVIII/4-W13/ID_57.pdf

[11] (2011) Ross Mason. "Introducing integration PaaS (iPaaS)." Dostopno na: http://blogs.mulesoft.org/introducing-integration-paas-ipaas/

[12] Tim Mather, Subra Kumaraswamy, Shahed Latif, *Cloud Security and Privacy*. Sebastopol: O'Reilly Media, 2009, pogl. 2.

[13] (2011) The National Institute for Standards and Technology. "The NIST Definition of Cloud." Dostopno na: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[14] (2011) Microsoft. "Standard Cloud Taxonomies and Windows Azure." Dostopno na: http://blogs.msdn.com/b/cclayton/archive/2011/06/07/standard-cloud-taxonomies-and-windows-azure.aspx

[15] (2011) Microsoft. "Windows Azure." Dostopno na: http://www.microsoft.com/windowsazure/

[16] (2011) Montclair Advisors. "Dreamforce 10: Force.com Becomes a PaaS Suite." Dostopno na: http://montclairadvisors.com/blog/2010/12/dreamforce-10-forcecom-becomes-a-paas-suite/

[17] (2011) Morgan Stanley Research. "Cloud Computing Takes Off." Dostopno na: http://www.morganstanley.com/views/perspectives/cloud_computing.pdf

[18] (2011) MuleSoft. "What is iPaaS? Gartner Provides a Reference Model." Dostopno na: http://www.mulesoft.com/what-is-ipaas-gartner-provides-reference-model

[19] (2011) Keith Pijanowski. "The Many Flavors of Cloud Computing." Dostopno na: http://www.keithpij.com/Home/tabid/36/EntryId/24/Default.aspx

[20] (2011) Keith Pijanowski. "Understanding Public Clouds: IaaS, PaaS, & SaaS." Dostopno na: http://www.keithpij.com/Home/tabid/36/EntryID/27/Default.aspx

[21] (2011) Archie Reed, Stephen G. Bennet, *Silver Clouds, Dark Linings: A Concise Guide to Cloud Computing*, Prentice Hall, 2010, pogl. 1. Dostopno na: http://ptgmedia.pearsoncmg.com/images/9780131388697/samplepages/013138869X.pdf

[22] (2011) Salesforce.com. "Force.com." Dostopno na: http://www.salesforce.com/platform/

[23] (2011) Salesforce.com. "What is PaaS." Dostopno na: http://www.salesforce.com/paas/

[24] (2011) The Eclipse Foundation. "Eclipse Helios." Dostopno na: http://eclipse.org/helios/

[25] (2011) Wikipedia. "Cloud computing." Dostopno na: http://en.wikipedia.org/wiki/Cloud_computing

[26] (2011) Wikipedia. "Platform as a service." Dostopno na: http://en.wikipedia.org/wiki/Platform_as_a_service