Referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka

Robert Dukarić

doktorska disertacija

predana

Fakulteti za računalništvo in informatiko

kot del izpolnjevanja pogojev za pridobitev naziva

doktor znanosti

s področja

računalništva in informatike

Ljubljana, 2014

XLAB NOT IDLE

Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

Referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka

Robert Dukarić

DOKTORSKA DISERTACIJA
PREDANA
FAKULTETI ZA RAČUNALNIŠTVO IN INFORMATIKO
KOT DEL IZPOLNJEVANJA POGOJEV ZA PRIDOBITEV NAZIVA
DOKTOR ZNANOSTI
S PODROČJA
RAČUNALNIŠTVA IN INFORMATIKE

Ljubljana, 2014

IZJAVA

Izjavljam, da sem avtor doktorske disertacije z naslovom Referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka, ki sem jo izdelal samostojno pod vodstvom mentorja in da slednje ne vsebuje materiala, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali na drugem visokošolskem zavodu, razen v primerih, kjer so navedeni viri.

— Robert Dukarić —

september 2014

Oddajo so odobrili

dr. Matjaž B. Jurič
redni profesor za računalništvo in informatiko
MENTOR IN ČLAN OCENJEVALNE KOMISIJE

dr. Patricio Bulić
izredni profesor za računalništvo in informatiko
PREDSEDNIK OCENJEVALNE KOMISIJE

dr. Marjan Krisper
izredni profesor za računalništvo in informatiko
ČLAN OCENJEVALNE KOMISIJE

dr. Roman Trobec
izredni profesor za računalništvo in informatiko
ZUNANJI ČLAN OCENJEVALNE KOMISIJE
Institut Jožef Štefan

PREDHODNA OBJAVA

Izjavljam, da so bili rezultati obravnavane raziskave predhodno objavljeni/sprejeti za objavo v recenzirani reviji ali javno predstavljeni v naslednjih primerih:

[1] R. Dukaric, M. B. Juric. Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems*, 29(5):1196-1210, 2013. doi: 10.1016/j.future.2012.09.006

[2] R. Dukaric, M. B. Juric. A taxonomy and survey of infrastructure-as-a-service systems. *Lecture notes on information theory*, 1(1):29-33, 2013.

[3] R. Dukaric, M. B. Juric. Migracija obstoječih aplikacij na platforme za računalništvo v oblaku. *Uporabna informatika*, 19(3):136-146, 2011.

[4] R. Dukaric, M. B. Juric. A unified architecture of IaaS cloud solutions. *Proceedings of the 1st International Conference on CLoud Assisted ServiceS*, str. 28–34, Bled,2012.

[5] R. Dukaric, M. B. Juric. Migracija obstoječih rešitev v oblak. *18. konferenca Dnevi slovenske informatike*, str. 1–6, Portorož,2011.

[6] R. Dukaric, M. B. Juric. Prednosti in slabosti uporabe računalništva v oblaku v javni upravi. *Informatika kot gonilo razvoja javne uprave : zbornik konference Informatika v javni upravi 2010*, str. 1–11, Brdo pri Kranju,2010.

V recenzijo sta bila poslana še naslednja članka:

[7] R. Dukaric, R. Buyya, M. B. Juric. Compensation handling in cloud orchestrators. *Future Generation Computer Systems*.

[8] R. Dukaric, M. B. Juric. Cloud-specific enrichment of contemporary container-based platforms. *Journal of Systems and Software*.

Potrjujem, da sem pridobil pisna dovoljenja vseh lastnikov avtorskih pravic, ki mi dovoljujejo vključitev zgoraj navedenega materiala v pričujočo disertacijo. Potrjujem, da zgoraj navedeni material opisuje rezultate raziskav, izvedenih v času mojega podiplomskega študija na Univerzi v Ljubljani.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*
Robert Dukarić
Referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka

POVZETEK

Računalništvo v oblaku ponuja nov način dostopa do infrastrukture, platforme in aplikacij v obliki storitev. Trenutno je v svetu IT prisotna velika raznolikost platform IaaS (Infrastructure as a Service) in PaaS (Platform as a Service), kar predstavlja precejšnjo oviro pri gradnji in izvajanju kompleksnih aplikacij v infrastrukturnih in platformskih oblakih. Posledično so potrebe po enotnem in celovitem referenčnem modelu posameznega sloja ter modelu njune integracije zelo velike. Iz tega razloga je izhodišče predlagane doktorske teme izdelava referenčnega modela integracije infrastrukturnih in platformskih nivojev računalniškega oblaka, ki ga sestavljajo trije komplementarni modeli: (1) arhitekturni model IaaS, (2) model obogatitve sodobnih izvajalnih platform s specifikami oblaka ter (3) model upravljanja kompenzacij v orkestracijskih platformah oblaka.

Ena izmed vidnejših pomanjkljivosti IaaS je njeno izrazito pomanjkanje skupnega izhodišča za analizo, primerjavo ter evalvacijo njenih implementacij, saj trenutno ne obstaja enotna in celovita taksonomija ter arhitekturno ogrodje sloja IaaS. Iz tega razloga predlagamo arhitekturni model IaaS, ki ga sestavljata taksonomija in arhitekturno ogrodje IaaS. Za ovrednotenje klasifikacije predlaganega modela naredimo sistematični pregled in raziskavo številnih sistemov IaaS ter jih v okviru raziskovalnega dela preslikamo na predlagano taksonomijo. Nato definiramo arhitekturno ogrodje IaaS, ki temelji na enotni taksonomiji, podamo podroben opis posameznega sloja in definiramo odvisnosti med sloji in komponentami. Da bi pokazali učinkovitost in uporabnost predlaganega arhitekturnega modela IaaS, izvedemo ovrednotenje modela na številnih realnih projektih. V ta namen izvedemo celovito analizo najpomembnejših komercialnih in odprto-kodnih produktov IaaS. Rezultati evalvacije pokažejo opazne razlike pri funkcijski podpori med komercialnimi in odprto-kodnimi platformami IaaS, bistvene pomanjkljivosti pomembnih arhitekturnih komponent v smislu izpolnjevanja

obljub infrastrukturnih oblakov ter uporabnost predlaganega arhitekturnega modela v realnem svetu.

Za zagotovitev neprekinjenega izvajanja aplikacij v oblakih IaaS in PaaS morata biti v sodobnih aplikacijskih platformah naslovljena dva izmed pomembnejših aspektov: elastičnost in nadzor. Skladno s tem preučimo vidike elastičnosti in nadzora računalniških oblakov ter zasnujemo model obogatitve sodobnih izvajalnih platform tako, da definiramo generičen sistem parametrov za izvajanje aplikacij v oblakih IaaS in PaaS. Generičen sistem parametrov strukturiramo v dve skupini parametrov: parametri za nadzor in parametri elastičnosti. Obe skupini parametrov predstavljata platformsko-neodvisne parametre, ki ju uporabimo za preslikavo v platformsko-specifične programske direktive in politike. S preslikavo sistema parametrov v platformo Java EE želimo obogatiti sodobne izvajalne platforme z deficitnimi zmožnostmi oblaka (tj. elastičnost in nadzor) ter pokazati, da preslikava skrije kompleksnost spodaj ležeče infrastrukture sistemov, medtem ko administratorjem aplikacij in razvijalcem ponuja večji nadzor in upravljanje nad njihovimi aplikacijami.

Ena izmed pomembnejših lastnosti uspešne postavitve aplikacij v oblakih predstavlja orkestracija virov v oblaku. Orkestracija virov v oblaku opisuje avtomatizirano ureditev, koordinacijo in upravljanje kompleksnih procesov oblaka in je realizirana z orkestracijo delovnih tokov. Da bi omogočili zanesljivo orkestracijo delovnih tokov, ki je odporna na napake, je pomembno vzpostaviti učinkovit mehanizem upravljanja s kompenzacijami. Skladno s tem vpeljemo model upravljanja kompenzacij v orkestracijsko arhitekturo oblaka, pri čemer izvedemo arhitekturno analizo obstoječih sistemov za delovne tokove specifične za oblak, definiramo generični pristop za upravljanje kompenzacij, ki je aplikabilen nad orkestratorji oblaka, ter predlagamo nov algoritem Compensation Activities Search (CAS) za preiskovanje delovnih tokov orkestratorjev oblaka (tj. usmerjenih grafov) in iskanje primernih aktivnosti, ki jih je potrebno kompenzirati. Na koncu predstavimo primer uporabe adaptiranja BPMN 2.0 za orkestracijo opravil, specifičnih za oblak, ter razvijemo sistem PoC (Proof of Concept), s katerim prikažemo efektivnost in izvedljivost predlaganega pristopa.

Ključni rezultati doktorske disertacije so (1) arhitekturni model IaaS, ki definira enotno taksonomijo in arhitekturno ogrodje IaaS, podaja podroben opis slojev in definira odvisnosti med sloji in komponentami, (2) model obogatitve sodobnih izvajalnih platform s specifikami oblaka, ki analizira vidike elastičnosti in nadzora računalniških oblakov in definira generičen sistem parametrov za izvajanje aplikacij v oblakih IaaS

in PaaS ter (3) model upravljanja kompenzacij v orkestracijskih platformah oblaka, ki analizira arhitekturo obstoječih orkestratorjev oblaka, definira generični pristop za upravljanje kompenzacij ter vpeljuje nov algoritem CAS za preiskovanje delovnih tokov orkestratorjev oblaka. Z vpeljavo treh komplementarnih modelov v krovni referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka smo dosegli vse cilje, ki smo jih zastavili v doktorski disertaciji.

Ključne besede: računalništvo v oblaku, taksonomija, arhitekturno ogrodje, Java EE, orkestracija oblaka, upravljanje kompenzacij

University *of Ljubljana*
Faculty *of Computer and Information Science*
Robert Dukarić
Reference model for the integration of infrastructure- and platform-level clouds

ABSTRACT

Cloud computing is revolutionizing the IT industry by enabling access to infrastructure, platform and applications as services. Due to the vast diversity of existing IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) clouds, the development and execution of contemporary cloud-based applications is significantly hindered. Therefore, the goal of this thesis is to design a reference model for the integration of infrastructure- and platform-level clouds, which we structure into three models: (1) architectural model of IaaS, (2) model for cloud-specific enrichment of contemporary container-based platforms and (3) model for compensation handling in cloud orchestrators.

There is an evident deficiency of mechanisms for analysis, comparison and evaluation of IaaS cloud implementations, since no unified taxonomy or reference architecture is available. In this article, we propose an architectural model of IaaS, which consists of a unified taxonomy and an IaaS architectural framework. We survey various IaaS systems and map them onto our taxonomy to evaluate the classification. We then introduce an IaaS architectural framework that relies on the unified taxonomy. We provide detailed description of each layer and define dependencies between the layers and components. Finally, we evaluate the proposed IaaS architectural framework on several real-world projects, while performing a comprehensive analysis of the most important commercial and open-source IaaS products. The evaluation results show notable distinction of feature support and capabilities between commercial and open-source IaaS platforms, significant deficiency of important architectural components in terms of fulfilling true promise of infrastructure clouds, and real-world usability of the proposed taxonomy and architectural framework.

Elasticity and cloud-control are two most important aspects that have to be addressed in order to seamlessly run applications in IaaS and PaaS clouds. Hence, we have

investigated elasticity and cloud-control capabilities of contemporary container-based platforms. We have design a model for cloud-specific enrichment of contemporary container-based platforms, while defining a generic parameter system for executing applications in IaaS and PaaS clouds, which we have structured around two parameter groups: control parameters and elasticity parameters. Both groups present platform-independent parameters, which we use to derive platform-specific programming directives and policies, thus achieving cloud-specific enrichment with deficient elasticity and cloud-control capabilities. We have shown that applying such metadata system to Java EE platform provides application administrators and application developers with greater control and manageability of their application.

Cloud orchestration is one of the most important mechanisms of a successful application deployment in the cloud. Cloud orchestration describes the automated arrangement, coordination, and management of complex cloud systems, middleware and services, and is realized by orchestrating workflows. To enable reliable and fault-tolerant cloud orchestration, it is important to establish an effective compensation handling mechanism. In fact, compensation handling principles have so far not been included in cloud orchestration platforms and had to be performed manually, by implementing the reversal operations of already completed activities in one of the upstream activities or within an error handler, thus preventing workflow designers to compose more error-prone workflows in an efficient way. In this paper we introduce a model for compensation handling support in cloud orchestrators while performing an architectural analysis of existing cloud-specific workflow systems, defining a generic approach for compensation handling applicable to cloud orchestrators, and proposing a novel algorithm called Compensation Activities Search (CAS) for traversing a cloud orchestration workflow and finding appropriate activities to be compensated. Finally, we present a use case of adapting BPMN 2.0 to orchestrate cloud-specific tasks and develop a proof of concept system to show the effectiveness and feasibility of our proposed approach.

Key results of the dissertation are (1) an architectural model of IaaS, which defines a unified taxonomy and an IaaS architectural framework, provides detailed description of each layer and defines dependencies between the layers and components, (2) a model for cloud-specific enrichment of contemporary container-based platforms, which investigates elasticity and cloud-control capabilities of contemporary container-based platforms and defines a generic parameter system for executing applications in IaaS and PaaS clouds, and (3) a model for compensation handling support in cloud

orchestrators, which performs an architectural analysis cloud orchestrators, defines a generic approach for compensation handling applicable to cloud orchestrators, and introduces a novel algorithm CAS for traversing a cloud orchestration workflow. With the introduction of three complementary models into the parent reference model for the integration of infrastructure- and platform-level clouds, we achieved all the goals we have set in this doctoral dissertation.

Key words: cloud computing, taxonomy, architectural framework, Java EE, cloud orchestration, compensation handling

ZAHVALA

Iskreno se zahvaljujem svojim staršem, ki sta mi omogočila študij ter me vseskozi podpirala, tako na zasebni, kot tudi na poklicni poti. Za pomoč pri pisanju disertacije se zahvaljujem mentorju prof. dr. Matjažu B. Juriču ter vsem sodelavcem v laboratoriju. Posebna zahvala gre tudi bratu, sestri ter prijateljem, ki so me nesebično spodbujali in mi stali ob strani. Operacijo delno financira Evropska unija, in sicer iz Evropskega socialnega sklada.

— Robert Dukarić, Ljubljana, september 2014.

KAZALO

Povzetek *i*

Abstract *v*

Zahvala *ix*

1 Uvod *3*

1.1 Motivacija . . . 4
1.2 Prispevki k znanosti . . . 9
1.3 Metodologija . . . 12
1.4 Struktura disertacije . . . 15

2 Osnovni koncepti in ozadje *19*

2.1 Uvod . . . 20
2.2 Računalništvo v oblaku . . . 21
2.2.1 Definicija . . . 21
2.2.2 Lastnosti . . . 23
2.2.3 Storitveni modeli . . . 27
2.2.4 Namestitveni modeli . . . 28
2.2.5 Izhodna strategija . . . 30
2.2.6 Programski modeli . . . 30
2.3 Zagotavljanje SLA-jev v oblaku . . . 31
2.3.1 Predstavitev vsebine dokumenta SLA . . . 33
2.3.2 Ogrodja dogovora na ravni storitve - SLA . . . 34
2.4 Migracija v oblak . . . 41

2.4.1 Tipi migracije v oblak 41
2.4.2 Prednosti in slabosti migracije v oblak 44
2.5 Orkestracija storitev oblaka 46

3 Arhitekturni model IaaS *51*
3.1 Uvod 52
3.2 Taksonomija IaaS 53
3.3 Arhitekturno ogrodje IaaS 55
3.4 Arhitekturni sloji 57
3.4.1 Sloj abstrakcije virov 57
3.4.2 Sloj jedrnih storitev 59
3.4.3 Podporni sloj 61
3.4.4 Sloj varnosti 62
3.4.5 Sloj upravljanja 63
3.4.6 Nadzorniški sloj 67
3.4.7 Storitve z dodano vrednostjo 68
3.5 Funkcijske odvisnosti 71
3.6 Evalvacija obstoječih arhitektur IaaS 74

4 Obogatitev sodobnih izvajalnih platform s specifikami oblaka *83*
4.1 Uvod 84
4.2 Pregled obstoječih izvajalnih platform 85
4.3 Predlagan sistem parametrov 88
4.3.1 Parametri za nadzor 91
4.3.2 Parametri elastičnosti 98
4.4 Evalvacija: preslikava parametrov v sistem metapodatkov Java EE . . 102
4.4.1 Metapodatki za nadzor 105
4.4.2 Metapodatki elastičnosti 109
4.5 Rezultati in diskusija 114

5 Upravljanje kompenzacij v orkestracijskih platformah oblaka *117*
5.1 Uvod 118
5.2 Upravljanje kompenzacij 118
5.3 Platforme za orkestracijo oblaka 122
5.4 Vpeljava kompenzacij v orkestracijske platforme oblaka 125

5.4.1 Arhitekturna analiza 125
5.4.2 Predlagana rešitev za upravljanje kompenzacij 134
5.5 Evalvacija 140
5.5.1 Razširitve BPMN 2.0 s specifikami oblaka 142
5.5.2 Preslikava na BPMN 2.0 145
5.5.3 Proof-of-concept 147

6 Sorodne raziskave in diskusija *155*
6.1 Arhitekturni model IaaS 156
6.2 Model obogatitve sodobnih izvajalnih platform s specifikami oblaka . 158
6.3 Upravljanje kompenzacij v orkestracijskih platformah oblaka 162

7 Zaključek *169*
7.1 Sklepi 170
7.2 Prispevki k znanosti 173
7.3 Prihodnje delo 174

Literatura *179*

"Do not go where the path may lead, go instead where there is no path and leave a trail."

— Ralph Waldo Emerson

I

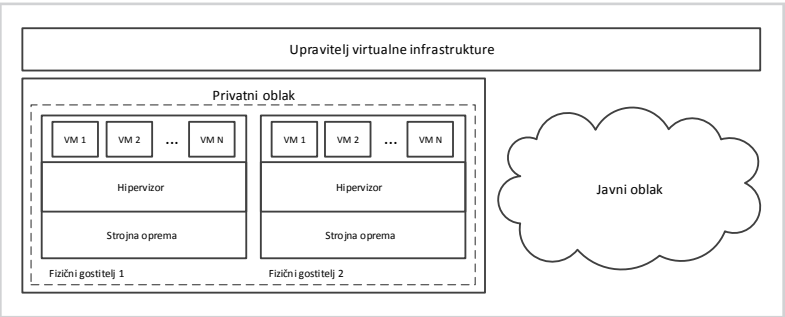
Uvod

1.1 Motivacija

Računalništvo v oblaku je nova paradigma, ki omogoča oskrbovanje različnih računalniških virov, in je običajno naslovljena iz treh temeljnih aspektov: infrastrukture kot storitve, platforme kot storitve ter programske opreme kot storitve. Zaradi hitre rasti omenjenega področja v okolju IT, se je pojavilo veliko število različnih definicij, ki so povzročile zmedo o tej paradigmi in zmožnostih, ki jih ponuja. Da bi zmanjšali te nejasnosti, so se v akademski literaturi pojavili poskusi definiranja podrobne ontologije računalništva v oblaku [1]. Iz podobnega razloga so se v akademskem svetu pojavile tudi nekatere nove definicije [2–8], ki v večini primerov temeljijo na študiji obstoječih literarnih del.

Obstajajo številni poskusi definiranja referenčne arhitekture računalništva v oblaku v industriji [9–11], različnih združenjih [12–15] ter v akademskem svetu [16–18]. T. Tung [16] je definiral referenčni model oblaka ter identificiral funkcionalnosti, ki upoštevajo arhitekturni sklad aplikacij računalniškega oblaka. Ravno tako je združil funkcionalnosti IT v sloje, da bi ločil temeljna področja oblaka. Čeprav predlagan model identificira potrebne komponente za arhitekturo oblaka, izkazuje izrazito pomanjkanje fokusa na infrastrukturnem nivoju, saj ne obravnava pomembnih storitev z dodano vrednostjo (angl. Value-added services), kot so orkestracija oblaka, migracija v času izvajanja ter podpora za HA (High Availability). Y. Sune et al. [17] so kombinirali upravljavske koncepte upravljanja storitev IT z arhitekturnim modelom upravljanja in spremljanja oblaka, vendar pri tem niso upoštevali karakteristik, kot sta varnost in nadzor oblaka. Podobno so N. Loutas et al. [18] predlagali semantično-podprto in odprto referenčno arhitekturo (RASIC) ter model za poenoten API oblaka, ki morebiti olajša menjavo ponudnika oblaka ter eliminira problematiko zaklepanja pri enem samem ponudniku (angl. Vendor Lock-In). Kljub temu predlagana referenčna arhitektura ne naslavlja upravljavskih in varnostnih aspektov, ki so pri računalniškem oblaku zelo pomembni. Na podoben način so nastale nekatere obetavne referenčne arhitekture s strani različnih združenj ter industrije:

- Referenčne arhitekture, ki so jih predlagali NIST (National Institute of Standards and Technology) [12], Distributed Management Task Force (DMTF) [13], Oracle [9] ter IBM [10] klasificirajo svoje arhitekturne komponente glede na različne akterje računalniškega oblaka.



Slika 1.1

Upravljanje oblakov z uporabo upravitelja virtualne infrastrukture.

- Arhitekturo, ki je osredotočena bolj na razslojenost (angl. Layer-based), so predlagali Cisco [11], Cloud Security Alliance (CSA) [14] in Internet Engineering Task Force (IEFT) [15].

Čeprav vse omenjene arhitekture (različnih združenj ter industrije) predstavljajo splošno-namenske modele oblaka, same ne naslavljajo komponent, ki jih mi uvrščamo pod storitve z dodano vrednostjo, in ne vpeljujejo odvisnosti med komponentami ter sloji. Ravno tako te arhitekture ne temeljijo na taksonomiji in nekatere izmed njih so specifične ponudniku (angl. Vendor Specific).

Infrastruktura kot storitev (angl. Infrastructure as a Service – IaaS) postaja najbolj uporabljen dostavni model v industriji, kot tudi akademiji, saj predstavlja koristno računalniško rešitev s preverjeno zmožnostjo zmanjševanja stroškov ter izboljšanja učinkovitosti in utilizacije virov. Glede na definicijo organizacije NIST [19], je IaaS definirana kot zmožnost oskrbovanja procesnih, diskovnih, omrežnih ter drugih temeljnih računalniških virov, pri čemer lahko uporabnik namešča in izvaja poljubno programsko opremo (tj. operacijske sisteme in aplikacije). Uporabnik pri tem ne vzdržuje in ne nadzira spodaj ležeče infrastrukture, temveč ima nadzor nad operacijskimi sistemi, shrambo in postavljenimi aplikacijami ter običajno omejen nadzor nad izbranimi omrežnimi komponentami (npr. požarni zid gostitelja) [20].

Ključen mehanizem, ki je zadolžen za orkestracijo, upravljanje ter agregacijo virtualnih virov iz več vozlišč (tj. strežnikov, shrambe ter omrežij), se imenuje upravitelj virtualne infrastrukture (angl. Virtual Infrastructure (VI) Manager) [21–23]. Nekateri viri uporabljajo termine, kot so "IaaS toolkits" [24, 25], "Virtual Infrastructure Engine (VIE)" [26, 27] ali "Virtual Execution Environment Manager (VEEM)" [28]. Takšna

ogrodja, ki omogočajo upravljanje oblakov, predstavljajo temelj za izgradnjo privatnih in hibridnih IaaS oblakov (slika 1.1). Kljub temu je prisotno izrazito pomanjkanje skupnega besednjaka za analizo, primerjavo ter evalvacijo implementacij IaaS, saj trenutno ne obstaja enotna in celovita referenčna arhitektura sloja IaaS. Večina današnjih rešitev IaaS je komercialnih, vendar obstaja tudi nekaj odprto-kodnih rešitev tako v akademiji, kot tudi v industriji. Zaradi vedno večjega interesa po adapciji ogrodij IaaS v organizacijah IT se je pojavila potreba po mehanizmih za celovito primerjavo in odločitvenih modelih arhitektur IaaS. Arhitekture IaaS so v literaturi že bile primerjane in analizirane [21, 29–38], vendar trenutno ne obstajajo modeli, ki bi primerjavo ter evalvacijo naslavljali skozi celoten sklad IaaS.

Platforma kot storitev (angl. Platform as a Service – PaaS) na drugi strani predstavlja izvajalno okolje za aplikacije v oblaku. Izvajalna okolja, ki se na nivoju PaaS največkrat pojavljajo so platforma .NET [39], Java EE [40], sklad LAMP (Linux, Apache, MySQL ter PHP) ter platforme SOA/BPM (Service-Oriented Architecture/Business Process Management) [41, 42]. Glede na definicijo inštituta NIST [19] PaaS predstavlja zmožnost nameščanja aplikacij na infrastrukturo oblaka, pri čemer so aplikacije implementirane s pomočjo programskih jezikov, knjižnic, storitev ter orodij ponujenih s strani ponudnika oblaka PaaS. V omenjenem primeru uporabniku ni potrebno vzdrževati in nadzirati spodaj ležeče infrastrukture oblaka, kot so omrežje, strežniki, operacijski sistemi in shramba, vendar ima večji nadzor nad postavljenimi aplikacijami [20].

V današnjem času razvijalci vedno pogosteje prepoznavajo potrebo po distribuiranih, transakcijsko podprtih in portabilnih aplikacijah, ki koristijo hitrost, varnost in zanesljivost strežniških tehnologij (angl. Server-Side Technology). Poslovno-informacijske aplikacije (angl. Enterprise Applications), ki ponujajo poslovno logiko, so centralno upravljane in so pogosto v interakciji z ostalo poslovno-informacijsko programsko opremo. V svetu informacijske tehnologije morajo biti takšne aplikacije načrtovane, zgrajene in nameščene z manj stroški, truda ter z večjo hitrostjo. S prihodom sodobnih platform, kot sta .NET in Java EE, je razvoj takšnih aplikacij postal bistveno lažji kot v predhodnih časih [43, 44]. Kljub temu izvajanje in razvoj poslovnih aplikacij temeljita na vsebnikih, ki niso arhitekturno zasnovani za izvajanje v okolju oblaka (IaaS in PaaS). Hitre spremembe na nivoju aplikacijskih strežnikov danes od aplikacij zahtevajo vpeljavo nekaterih implementacijskih (arhitekturnih) sprememb ter posledično vpeljavo novih metapodatkov (angl. Metadata Annotations) [45] oziroma razširitev obstoječih.

Čeprav je to področje še relativno slabo raziskano, se je že nekaj avtorjev [46–48] ukvarjalo z vpeljavo aspektov računalniškega oblaka v aplikacijsko platformo. Vsi omenjeni avtorji naslavljajo zgolj aspekt večnajemniškega modela (angl. Multi-tenancy), vendar ne upoštevajo dveh dodatnih karakteristik računalništva v oblaku: elastičnosti ter nadzorniških funkcij oblaka.

Sodobni aplikacijski strežnik, ki se izvaja v okolju oblaka IaaS in PaaS mora upoštevati tri temeljne koncepte: večnajemniški model, elastičnost ter nadzor (spremljanje) aplikacij. V kontekstu večnajemniškega modela je potrebno upoštevati različne nivoje večnajemniških modelov ter nekonvencionalno dejstvo, da si lahko več najemnikov deli/izmenjuje en sam izvajalni vsebnik. Podobno velja tudi na podatkovnem nivoju (npr. JPA – Java Persistence API [49], JDBC - Java Database Connectivity [50]), kjer si lahko več različnih najemnikov deli/izmenjuje tudi skupne tabele. Dodatno je potrebno upoštevati, da je večslojna aplikacija v okolju oblaka distribuirana med različnimi virtualnimi stroji, gručami ter celo med različnimi podatkovnimi centri. Pri tem mora biti izenačevanje obremenitve konfigurirano tako, da zna razporejati zahtevke v geografsko distribuiranem okolju oblaka. V primeru elastičnosti je potrebno nasloviti dva aspekta: ročno ter samodejno skalabilnost. Ročna skalabilnost je dosežena z ročno intervencijo uporabnika oz. administratorja oblaka. Instance vsebnikov bodo v tem primeru programsko dodane ter odstranjene iz okolja oblaka, medtem ko bo podobno veljalo za instance sistema za upravljanje s podatkovno bazo. V primeru samodejne skalabilnosti, vsebniki ter instance podatkovne baze skalirajo samodejno - glede na uporabniško definirana pravila za skalabilnost - ki se preverjajo v določenem časovnem intervalu in v primeru, ko je pravilom zadoščeno, sprožijo ustrezne akcije. Takšna pravila so običajno definirana v dokumentu XML, ki mora biti skladen s predpisano shemo. Vidik nadzora aplikacij, ki se izvajajo v oblaku, temelji na mehanizmih, ki omogočajo aktiven nadzor in spremljanje aplikacije kot celote. Različni sistemi za spremljanje (angl. Monitoring) so zmožni pobirati številne metrike iz izvajalnega okolja in infrastrukture oblaka, s pomočjo katerih se izračunajo višje-nivojski parametri SLA (Service Level Agreement), kot je na primer razpoložljivost. V izogib morebitnim kršitvam je aktivno spremljanje in ustrezno odzivanje na parametre SLA nujno potrebno [51–53].

Orkestracija oblaka predstavlja skupno ime za avtomatizirano ureditev, koordinacijo in upravljanje sistemov, skladov programske opreme in praks, kar omogoča (1) poljubno avtomatizacijo, (2) integracijo navzkrižnih tehnologij, ter (3) izpolnjevanje storitve-

nih zahtev (angl. Service Request Fulfilment) od začetka do konca [54, 55]. Ravno tako omogoča upravljanje kompleksnih, distribuiranih procesov in ponuja orodja za kombiniranje programske opreme, strojne opreme, in ročnih procesov v brezhibno delujoč sistem. Takšna orodja nam omogočajo povezovanje in avtomatizacijo delovnih tokov, ki lahko funkcionirajo posamezno ali zaganjajo nove delovne tokove [56]. Upoštevajte, da sta pojma orkestracija in avtomatizacija večkrat zamenljiva pojma. Avtomatizacija se nanaša na ponavljajoči se proces, ki ga lahko programiramo, da funkcionira brez ročne intervencije, medtem ko je orkestracija širši termin in kombinira več avtomatiziranih procesov v delovni tok (angl. Workflow) ali nabor delovnih tokov. Delovni tok si lahko predstavljamo kot predlogo za definiranje zaporedja računskih in/ali podatkovnih procesnih opravil, ki so potrebna za upravljanje poslovnih, tehničnih ter znanstvenih procesov [57]. Ker storitvena orkestracija oblaka predstavlja hitro razvijajoče se raziskovalno področje, ne obstajajo dobro definirani standardi delovnih tokov, ki bi lahko učinkovito kljubovali poslovnim in tehničnim kompleksnostim. Pomembna težava pri upravljanju delovnih tokov oblaka je v obnovitvi napak (angl. Failure Recovery). Cilj obnovitve napak je privesti neuspešne delovne tokove nazaj v semantično sprejemljivo stanje, tako da je lahko vzrok napake identificiran, težava odpravljena in izvajanje nadaljevano. Da bi uspešno spravili delovni tok nazaj v semantično sprejemljivo stanje, je potrebno kompenzirati že zaključene aktivnosti, dokler ne preidemo v želeno stanje [58].

V zadnjih letih je bilo veliko pozornosti posvečene platformam za orkestracijo oblaka v akademiji, kot tudi industriji. Skoraj vsa večja podjetja IT, npr. Microsoft [56], VMware [59], BMC [60], Cisco [61], HP [62] ter IBM [63], so razvila lastniške rešitve orkestratorjev oblaka, ki omogočajo avtomatizacijo storitev oblaka, od začetka do konca. Kljub temu so ti pristopi specifični ponudnikovi tehnologiji in ne vpeljujejo semantike upravljanja kompenzacij v njihove platforme delovnih tokov. Še več, principi upravljanja s kompenzacijami so doslej morali biti izvedeni ročno tako, da so razvijalci implementirali reverzne operacije (angl. Reversal Operations) že zaključenih aktivnosti v eni izmed prihajajočih (angl. Upstream) aktivnostih ali znotraj upravljavca napak (angl. Error Handler). Na tak način načrtovalci delovnih tokov doslej niso mogli oblikovati delovnih tokov, ki bi bili na učinkovit način bolj odporni na napake. Upravljanje kompenzacij je en izmed mehanizmov za upravljanje napak v izvajajočih se instancah delovnih tokov. Njihov cilj je povrniti (angl. Revert) učinek operacij, ki so bile uspešno zaključene, preden je prišlo do napake, in sicer z razlogom, da bi

prešli nazaj v konsistentno stanje. Opazen primer takšne tehnologije delovnih tokov, ki podpirajo upravljanje s kompenzacijami, je Business Process Model and Notation (BPMN). Uporablja se za definiranje poslovnih delovnih tokov in je postal "de facto" standard za modeliranje poslovnih procesov [64]. Trenutno BPMN in ostale tehnologije poslovnih delovnih tokov (npr. BPEL [65], YAWL [66] in Windows Workflow Foundation [67]) ne podpirajo modeliranje in izvajanje delovnih tokov specifičnih za oblak, vendar v večini primerov ponujajo podporo modeliranja in izvajanja semantike upravljanja kompenzacij. Razširitve BPMN 2.0 tako predstavljajo priložnost za zasnovo tehnološko-neodvisne arhitekture, ki je specifična za oblak in podpira upravljanje kompenzacij tako v poslovnih, kot tudi tehničnih procesnih domenah. Kombiniranje konceptov BPMN in orkestratorjev oblaka je smiselno, saj ponuja oblikovalcem delovnih tokov in razvijalcem najboljše iz obeh svetov.

1.2 Prispevki k znanosti

Izhodišče predlagane doktorske teme je izdelava referenčnega modela integracije IaaS in PaaS nivojev računalništva v oblaku. Trenutno je v industriji IT prisotna velika raznolikost platform IaaS in PaaS. Posledično so potrebe po enotnem in celovitem referenčnem modelu posameznega sloja ter modelu njune integracije zelo velike. Znotraj predstavljenega področja lahko identificiramo naslednje raziskovalne probleme:

- Neenoten pogled na arhitekturi IaaS in PaaS v distribuiranih okoljih računalniškega oblaka.
- Obstoječa ogrodja za analizo, primerjavo ter evalvacijo implementacij IaaS ne naslavljajo vseh ključnih aspektov nivoja računalniškega oblaka IaaS.
- Obstoječe referenčne arhitekture oblaka so zelo splošne in se ne osredotočajo na specifične storitvene modele. Kot posledica so nastajajoče arhitekture IaaS zelo heterogene.
- Sodobni izvajalni vsebniki niso arhitekturno zasnovani za izvajanje aplikacij v okoljih IaaS in PaaS. Obstoječe raziskave se v glavnem osredotočajo na reševanje problemov večnajemniškega modela [46–48].

- Funkcije elastičnosti in nadzora, ki bi izkoriščale prednosti oblaka in s tem bistveno izboljšale upravljanje in nadziranje aplikacij v oblaku IaaS in PaaS, v sodobnih izvajalnih platformah niso obravnavane.
- Obstoječe izvajalne platforme ne podpirajo naslednjih funkcionalnosti: (1) pridobivanje metrik IaaS in PaaS v javansko kodo, (2) zmožnost omejevanja izvajanja logičnih delov aplikacij med različnimi podatkovnimi centri ter (3) zmožnost obveščanja ob ustreznih dogodkih (npr. instanciranje novega spletnega vsebnika ali kršitev SLA).
- Trenutne verzije izvajalnih platform, ki temeljijo na vsebnikih, ne omogočajo nadziranja elastičnosti (tj. horizontalne skalabilnosti) aplikacij računalniškega oblaka. In sicer v programski kodi ni mogoče določiti začetnega/maksimalnega števila instanc spletnih vsebnikov ter definirati pravil za samodejno elastičnost aplikacij, ki se izvajajo v sistemih IaaS in PaaS.
- Obstoječe tehnologije orkestratorjev oblaka ne ponujajo podpore za obravnavo kompenzacij, ki bi bile uporabne v številnih poslovnih in tehničnih scenarijih delovnih tokov. Podpora obravnave kompenzacij bi načrtovalcem in razvijalcem delovnih tokov ponujala možnost razvoja delovnih tokov, ki bi bili odpornejši na napake.

Na podlagi predhodno izpostavljenih pomanjkljivosti v doktorski disertaciji pozornost namenimo definiranju treh arhitekturnih modelov: (1) arhitekturnega modela IaaS, (2) modela obogatitve sodobnih izvajalnih platform s specifikami oblaka ter (3) modela upravljanja kompenzacij v orkestracijskih platformah oblaka. Na podlagi opredelitve raziskovalnega problema so naši izvirni prispevki k znanosti oz. cilji predlaganega doktorskega dela naslednji:

- Definirati referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka ter v okviru tega modela definirati tri komplementarne modele (arhitekturni model IaaS, model obogatitve sodobnih izvajalnih platform s specifikami oblaka ter model upravljanja kompenzacij v orkestracijskih platformah oblaka). Referenčni model integracije (1) naslavlja integracijske izzive izvajalnih platform in infrastrukturnih oblakov, (2) definira stične komponente oz. sinergije predlagane integracije ter (3) vpeljuje deficitne zmožnosti v krovni, integracijski model oblaka.

- Definirati arhitekturni model IaaS, ki ga sestavljata med seboj povezana taksonomija IaaS ter arhitekturno ogrodje IaaS. Namen taksonomije je identificirati in klasificirati fundamentalne komponente IaaS v urejene kategorije/sloje ter uporabiti taksonomijo za oblikovanje arhitekturnega ogrodja IaaS. Namen arhitekturnega ogrodja IaaS pa je (1) organizirati konceptualne sloje/komponente v arhitekturno ogrodje, (2) ponuditi skupno izhodišče za analizo, primerjavo in evalvacijo implementacijskih arhitektur IaaS ter (3) definirati odvisnosti med posameznimi sloji in komponentami.

- Definirati model obogatitve sodobnih izvajalnih platform (tj. sistem parametrov) za izvajanje aplikacij v oblakih IaaS in PaaS ter ga organizirati v dve skupini: parametri za nadzor ter parametri za elastičnost. Predlagan formalni sistem transformira aplikacijske zahteve, ki so opredeljene s strani administratorja aplikacije ali razvijalca, v operacije nad spodaj ležečimi viri IaaS (z uporabo politik in programskih direktiv).

- Iz abstraktnega modela parametrov izpeljati model metapodatkov za platformo Java EE. Preslikava vključuje (1) deklaracijo novih anotacijskih metapodatkov, (2) razširitev globalnega postavitvenega deskriptorja (*application.xml*), (3) razširitev postavitvenega deskriptorja *ejb-jar.xml* in (4) razširitev postavitvenega deskriptorja *web.xml*. Predlagane deklaracije in razširitve naslavljajo aspekte elastičnosti ter nadzora aplikacij v oblaku.

- Vpeljati celovit model upravljanja kompenzacij v orkestratorje oblaka, pri čemer (1) izvedemo arhitekturno analizo obstoječih sistemov za delovne tokove specifične za oblak, (2) definiramo generični pristop za upravljanje kompenzacij, ki je aplikabilen nad orkestratorji oblaka ter (3) predlagamo nov algoritem Compensation Activities Search (CAS) za preiskovanje delovnih tokov orkestratorjev oblaka (tj. usmerjenih grafov) in iskanje primernih aktivnosti, ki jih je potrebno kompenzirati.

- Preslikati model upravljanja kompenzacij v razširjeno specifikacijo BPMN 2.0, ki vpeljuje nove tipe opravil s specifikami oblaka.

Del rezultatov doktorske disertacije je vključen v naslednje znanstvene članke [68–75]. Z uvedbo referenčnega modela integracije infrastrukturnih in platformskih nivo-

jev računalniškega oblaka lahko omogočimo vpeljavo treh komplementarnih modelov, kjer:

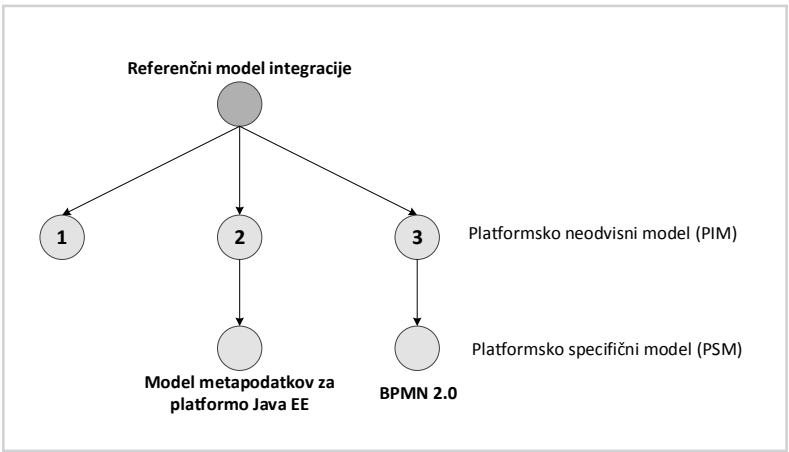
1. *arhitekturni model IaaS* izboljša razumevanje nivoja IaaS in omogoči lažje sprejemanje tehnoloških odločitev,
2. *model obogatitve sodobnih izvajalnih platform s specifikami oblaka* omogoča administratorjem in razvijalcem aplikacij večji nadzor in boljšo upravljivost aplikacij v oblaku,
3. *model upravljanja kompenzacij v orkestracijskih platformah oblaka* povečuje učinkovitost pri oblikovanju robustnih delovnih tokov, ki so odpornejši na napake.

Na podlagi izpostavljenega raziskovalnega problema je glavni cilj disertacije predlagati referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka ter v okviru tega modela definirati tri komplementarne modele. Slednja dva modela (2)(3), ki sta neodvisna od platforme (angl. Platform Independent Model – PIM), nato v skladu s pristopom MDA (Model Driven Architecture) preslikamo v dva platformsko specifična modela (Platform Specific Model – PSM). In sicer model obogatitve sodobnih izvajalnih platform preslikamo v Java EE Annotation Metadata ter model upravljanja kompenzacij preslikamo v razširjeno specifikacijo BPMN 2.0, ki trenutno predstavlja eno izmed najbolj aktualnih notacij za implementacijo poslovnih procesov (slika 1.2).

1.3 Metodologija

Pričakovane izvirne prispevke doktorskega dela smo dosegli z uporabo naslednjih pristopov dela:

1. Analiza obstoječih znanstvenih dognanj in tehnoloških dosežkov:
   (a) Podrobna študija sodobnih izvajalnih platform ter preučitev obstoječih aplikacijskih strežnikov.
   (b) Pregled in klasifikacija obstoječih programskih direktiv/anotacij in politik.
   (c) Analiza področja IaaS, pri čemer se osredotočimo na študijo in praktične preizkuse najpomembnejših odprto-kodnih ter komercialnih tehnologij.



Slika 1.2

Referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka.

(d) Preučitev področja PaaS ter identifikacija najprimernejših izvajalnih okolij ter njihovih prednosti in pomanjkljivosti.

(e) Analiza sinergij obstoječih izvajalnih platform in sistemov IaaS ter preučitev možnosti integracije med njimi.

2. Identifikacija potrebnih zahtev za definiranje arhitekturnega modela IaaS:

   (a) Na podlagi analize arhitekturnih zasnov sistemov IaaS je izvedena identifikacija ključnih zahtev ter gradnikov arhitekturnega modela IaaS.

   (b) Definiranje modela odvisnosti med posameznimi komponentami ter modela odvisnosti med posameznimi višje-nivojskimi sloji.

3. Definiranje in verifikacija arhitekturnega modela IaaS:

   (a) Oblikujemo taksonomijo ter arhitekturno ogrodje IaaS, ki rešujeta prej omenjene odprte probleme.

   (b) Predlagan model verificiramo na izbranih IaaS platformah.

4. Definiranje modela obogatitve sodobnih izvajalnih platform (tj. sistem parametrov) za izvajanje aplikacij v oblakih IaaS in PaaS:

(a) Na podlagi arhitekturnega modela IaaS ter podrobne študije PaaS okolja identificiramo bistvene zahteve in elemente potrebne za obogatitev izvajalnih platform s specifikami oblaka.

(b) Zasnujemo model obogatitve sodobnih izvajalnih platform, pri čemer klasificiramo identificirane elemente modela ter jih organiziramo v dve skupini: parametri za nadzor ter parametri za elastičnost.

5. Izdelava abstraktnega modela metapodatkov v platformi Java EE:

   (a) S pomočjo analize obstoječih anotacijskih metapodatkov Java SE/EE iz abstraktnega modela parametrov izpeljemo model metapodatkov ter jih organiziramo v dve skupini (metapodatki za elastičnost ter metapodatki za nadzor).

   (b) Preslikava med sistemom parametrov (angl. Parameter System) in sistemom metapodatkov (angl. Metadata System) je realizirana na štiri načine:

      i. deklaracija novih anotacijskih metapodatkov,
      ii. razširitev novega globalnega postavitvenega deskriptorja (*application.xml*),
      iii. razširitev postavitvenega deskriptorja *ejb-jar.xml*,
      iv. razširitev postavitvenega deskriptorja *web.xml*.

6. Definiranje modela upravljanja kompenzacij v orkestracijskih platformah oblaka:

   (a) Izvedemo arhitekturno analizo obstoječih sistemov za delovne tokove specifične za oblak, pri čemer najprej izvedemo identifikacijo in kategorizacijo ključnih elementov orkestracijskih delovnih tokov ter na koncu v kategorizacijo umestimo kompenzacijske elemente.

   (b) Definiramo generični pristop za upravljanje kompenzacij, ki je aplikabilen nad orkestratorji oblaka ter razvijemo algoritem CAS za preiskovanje delovnih tokov orkestratorjev oblaka.

7. Evalvacija in validiranje predlaganih modelov na realnih primerih:

   (a) Referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka je ovrednoten na realnih projektih tipa R&D, implementacijah PoC (Proof of Concept) ter industrijskih projektih.

1.4 Struktura disertacije

Doktorska disertacija je organizirana v sedem poglavij. V poglavju 2 podamo osnovne koncepte in ozadje, kjer najprej opredelimo širše raziskovalno področje in izpostavimo danes odprte probleme na področju računalniških oblakov. Poglavje 3 opiše arhitekturni model IaaS, kjer predstavimo taksonomijo IaaS in arhitekturno ogrodje, ter na koncu naredimo evalvacijo obstoječih arhitektur IaaS. V poglavju 4 predstavimo model obogatitve sodobnih izvajalnih platform s specifikami oblaka. Najprej naredimo pregled nad obstoječimi izvajalnimi platformami, nato opišemo predlagan sistem parametrov, ki ga na koncu evalviramo v obliki preslikave na sistem metapodatkov Java EE. Poglavje 5 predstavi model upravljanja kompenzacij v orkestracijskih platformah oblaka. V poglavju 6 opišemo sorodne raziskave, poglavje 7 pa poda zaključek doktorske disertacije in predlaga izhodišče za nadaljnje raziskave.

"Life begins at the end of your comfort zone."
— Neale Donald Walsch

2

Osnovni koncepti in ozadje

2.1 Uvod

Arhitekturni razvoj informacijskih tehnologij je zaradi svoje dinamične narave precej raznolik in zaradi tega v nenehnem razvoju. V šestdesetih in sedemdesetih letih dvajsetega stoletja je bil prvi val računalnikov sestavljen iz velikih in dragih monolitnih strežnikov, ki si jih lahko predstavljamo kot prednike osrednjega računalnika (angl. Mainframe). V osemdesetih in devetdesetih letih istega stoletja je z razvojem osebnega računalnika (angl. Personal Computer – PC), z namenom zmanjšanja stroškov računalniške in omrežne infrastrukture ter s potrebo po večji agilnosti, arhitektura odjemalec-strežnik omogočila delitev aplikacije od strežnika. Namen delitve je bila podpora porazdeljenim odjemalcem, ki so izvajali vedno bolj napredne uporabniške vmesnike. Delitev je dodatno zmanjšala stroške, do katerih pride pri migraciji aplikacijskih bremen iz monolitnih strežnikov. V prvem desetletju enaindvajsetega stoletja so se podatkovni centri pričeli množično širiti. Zaradi neprestanih potreb po vedno večji računalniški moči je najem in vzdrževanje računalniških prostorov ter računalniško hlajenje postalo vse dražje. Posledično so se koncepti, kot sta na primer koristno računalništvo (angl. Commodity Computing) ter virtualizacija, začeli uveljavljati v svetu računalništva. Ob podpori računalništva v oblaku omenjeni koncepti še dodatno omogočijo samopostrežbo, spremljanje porabe virov/storitev ter vzpostavijo avtomatizirano in dinamično upravljanje virov. V odgovor k povečanju porazdeljenosti storitev, se je kot metodologija za integracijo in vodenje distribuiranih poslovnih sistemov pojavila SOA. Ta potreba obstaja še danes, saj uporabniki zahtevajo integracijo med javnimi, privatnimi ter lokalnimi (angl. In-House) storitvami [76].

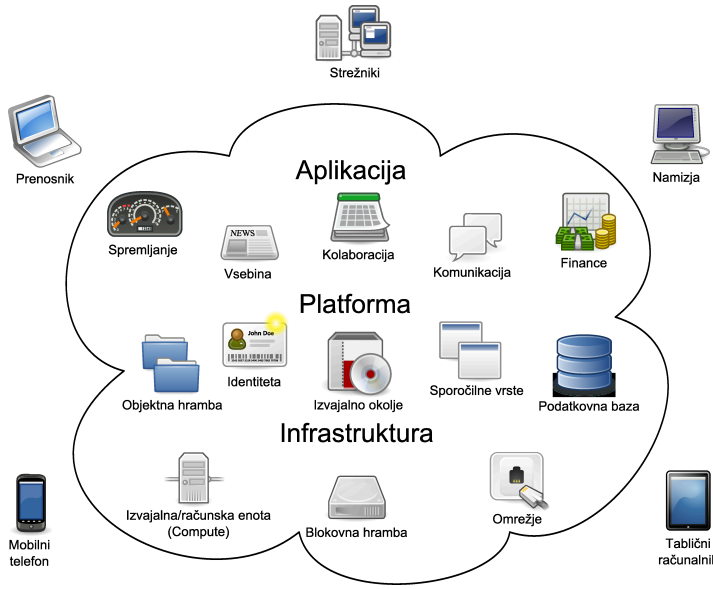
Evolucija namestitvenih modelov računalništva v oblaku se je začela z aplikacijskmi silosi, ki so postopoma prešli v tako imenovano mrežno računalništvo. Mrežno računalništvo se je kasneje razvilo v privatne oblake, ki pa so v današnji, informacijski dobi postali zelo uporabni in popularni. Najbolj obetavna topologija oblaka je ravno kombinacija privatnih in javnih oblakov in jih je potrebno razlikovati od preproste strežniške virtualizacije. Oblaki namreč omogočajo elastičnost dodeljevanja, avtomatizacije in dostopnost v obliki storitev. Medtem ko se virtualizacija osredotoča predvsem na konsolidacijo strežniških virov in posledično na boljšo izrabo ter večjo učinkovitost infrastrukture.

2.2 Računalništvo v oblaku

2.2.1 Definicija

Koncept računalništva v oblaku izvira iz šestdesetih let prejšnjega stoletja, ko je računalniški znanstvenik John McCarthy predlagal idejo računske moči, ki bi bila dostavljena kot javna storitev (angl. Public Utility) [77]. Seveda je računalništvo v oblaku resnično prišlo v veljavo šele z lansiranjem produkta Amazon EC2 leta 2006. Ker gre za relativno mlado področje, je iskanje enovite definicije računalništva v oblaku precej težavno. Na spletu lahko danes najdemo vsaj trideset različnih definicij o tej veji računalništva. Najbolj pogosto uporabljene in citirane definicije v strokovni literaturi bomo obravnavali v nadaljevanju.

Gartner Group, kot ena vodilnih analitskih in svetovalnih hiš s področja informacijske tehnologije, definira računalništvo v oblaku kot "slog računalništva, kjer so masivno skalabilne zmogljivosti IT, z uporabo internetnih tehnologij, kot storitev dostavljene zunanjim odjemalcem" [78]. Podjetji Forrester in Burton sta v svetu analitike IT in svetovanja prav tako zelo pomembna igralca. Slednji trdi, da je računalništvo v oblaku "nabor disciplin, tehnologij in poslovnih modelov, ki so kot storitev na voljo na zahtevo" [79]. Forrester pa definira računalništvo v oblaku kot "množico skalabilnih računalniških virov za gostovanje, ki omogočajo zaračunavanje glede na dejansko porabo" [80]. Wikipedia definira računalništvo v oblaku kot "računalništvo, pri katerem so dinamično razširljiva in pogosto virtualizirana računalniška sredstva na voljo kot storitev preko interneta" [81]. Ena izmed pogosteje uporabljenih definicij računalništva v oblaku je definicija inštituta NIST, ki deluje znotraj gospodarske zbornice Združenih držav Amerike. Razlog za široko uporabnost omenjene definicije predstavlja predvsem velik vpliv ameriške administracije na računalništvo v oblaku ter dejstvo, da so v letu 2012 namenili 70 milijard dolarjev letnega proračuna in s tem predstavljajo največjega uporabnika storitev IT na svetu [82]. NIST definira računalništvo v oblaku kot "model, ki omogoča omrežni dostop do deljenih računalniških virov (npr. omrežje, strežniki, shramba, aplikacije in storitve), ki so lahko rapidno oskrbovane (angl. Provisioned) in izdane (angl. Released) z minimalnim trudom vodstva oziroma interakcijo ponudnika storitve" [20]. UC Berkeley [5] pravi, da gre pri računalniškem oblaku za aplikacije, strojno opremo in sistemsko programsko opremo znotraj podatkovnih centrov, ki so v obliki storitev dostavljene znotraj medmrežja. Rezultati podrobne analize zgoraj navedenih definicij računalništva v oblaku kažejo na to, da se posamezne



Slika 2.1

Računalništvo v oblaku [81].

definicije vedno osredotočajo na dve različni področji. Prva predstavlja elastičnost in skalabilnost virov IT, medtem ko se druga osredotoča na dostopnosti storitve/virov v obliki storitev.

Poleg klasične virtualizacije je računalništvo v oblaku sestavljeno še iz avtomatizacije in orkestracije storitev ter večnajemniške zmožnosti. Ta paradigma je podprta z enotnimi, skalabilnimi bazeni strojne opreme, programsko opremo ter omrežnimi kapacitetami, da bi omogočili dinamično dostavo IT in aplikacijskih funkcionalnosti na zahtevo. Poslovni model računalništva v oblaku ponuja končnim uporabnikom različne stopnje fleksibilnosti v smislu uporabe storitev, zaračunavanja (angl. Billing) in samopostrežnega upravljanja ter oskrbovanja (angl. Provisioning). Model storitev v oblaku je lahko apliciran v naslednjih treh domenah (slika 2.1) [83]:

1. strežniki, shramba, omrežje ali infrastruktura kot storitev (IaaS),
2. okolje za razvoj aplikacij ali platforma kot storitev (PaaS),
3. funkcionalnosti poslovnih aplikacij ali programska oprema kot storitev (SaaS).

2.2.2 Lastnosti

V nadaljevanju opravimo pregled in opis ključnih lastnosti oz. karakteristik računalniških oblakov. Sem spadajo virtualizacija, avtomatizacija in orkestracija, federacija, elastičnost in skalabilnost, samopostrežba, večnajemniški model, merjenje storitev, konsolidacija in centralizacija, migracija virtualnih strojev in replikacija podatkov.

Virtualizacija

Strežniška virtualizacija omogoča izvajanje večjega števila slik OS/aplikacij (angl. Stack Images) na eni sami fizični strojni opremi in s tem predstavlja resnično vrednost računalniških oblakov. Hipervizorji (angl. Hypervisor) delijo (angl. Dividing) računalniške vire fizičnega strežnika na logične vire ali virtualne stroje, ki delujejo kot izolirana računalniška okolja in jih lahko po potrebi premikamo in kopirano med različnimi fizičnimi strežniki. Strežniška virtualizacija nam pomaga pri omejevanju strojne opreme s konsolidacijo aplikacij na manj fizičnih naprav. S tem izboljša utilizacijo strežniških virov, sprosti prostor v podatkovnem centru in zniža ceno operacij in energije.

Avtomatizacija in orkestracija

Orkestracija in avtomatizacija sta arhitekturno postavljeni nad infrastrukturo virtualizacije in ponujata mehanizem za hitro oskrbo in skaliranje okolja IT ter vzdrževanje jedra (angl. Core) infrastrukture oblaka. Sloj za orkestracijo povezuje zahtevke uporabnika z informacijami logičnega nivoja z namenom realizacije uporabniških zahtev. Sloj za orkestracijo na primer določi lokacijo začetne postavitve in sprejema odločitve za migracijo zahtevanih virov na osnovi pričakovane obremenitve in količine virov, ki so še na voljo v podatkovnem centru.

Federacija

Federacija v splošnem pomeni zbirko neodvisnih entitet, ki se obnašajo kot ena sama entiteta in pri tem ohranjajo individualno avtonomnost. V kontekstu računalništva v oblaku federacija pomeni združevanje več ponudnikov oblaka znotraj ene upravljavske

platforme. Za polno izpolnitev obljub računalništva v oblaku morajo obstajati tehnološke zmogljivosti za federacijo različnih podatkovnih centrov, tudi tistih, ki so v lasti drugih organizacij. Samo s pomočjo federacije in interoperabilnosti lahko ponudniki infrastrukture izkoristijo agregirano zmogljivost, z namenom zagotavljanja iluzije neskončnosti virov. Infrastrukturi, ki je skladna s to paradigmo, pravimo federiran oblak. Dodatno federacija iz ekonomskega vidika pozitivno vpliva na konkurenčnost in inovativnost, saj tudi manjšim in srednje velikim podjetjem omogoča lažji vstop na trg ponudnikov infrastrukture. Eden izmed vmesnikov, ki omogoča federacijo je CDMI (Cloud Data Management Interface). V zadnjih letih je bil standard CDMI razvit s strani vodilnih ponudnikov shrambe, uporabnikov in raziskovalcev tehnologij oblaka. Cilji omenjenega standarda so naslednji:

1. ponuditi standardni vmesnik, preko katerega lahko odjemalci komunicirajo,
2. ponuditi standardni pristop za dodajanje specifičnih funkcionalnosti ponudnika, ne da bi pri tem onemogočali kompatibilnost,
3. omogočiti standardizirane primere uporabe federacije med različnimi oblaki, ki predstavljajo možnost skaliranja preko omejitev enojnega podatkovnega sistema, delegirano shrambo, distribucijo shranjenih podatkov (angl. Content Delivery Network – CDN) in večnajemniški model (angl. Multitenancy).

Elastičnost in skalabilnost

Viri oblaka so lahko rapidno in elastično oskrbovani, v nekaterih primerih lahko skalirajo tudi samodejno. Zmožnosti, ki so na voljo za oskrbovanje, dajejo iluzijo neskončnosti virov in so na voljo uporabnikom, kadarkoli jih ti potrebujejo ter v poljubni količini [20]. Elastičnost izvira s področja fizike in ekonomije, vendar se je pred kratkim uveljavila tudi v kontekstu računalniških oblakov [84]. Elastičnost v kontekstu oblaka lahko definiramo kot stopnjo, do katere se lahko sistem prilagaja na spremembe delovnih obremenitev. V tem primeru sistem samodejno oskrbuje (dodaja in odvzema) vire tako, da se v vsakem danem trenutku razpoložljivi viri v čim večji meri ujemajo s trenutnimi zahtevami aplikacije. Elastičnost in skalabilnost sta obravnavani kot ključni karakteristiki, ki predstavljata diferenciator pred predhodnimi računalniškimi paradigmami, kot sta na primer mrežno računalništvo (angl. Grid Computing) ter storitveno računalništvo (angl. Utility Computing).

Elastičnost rešuje problem obremenitvenih konic sistema in predstavlja enega izmed bistvenih karakteristik računalništva v oblaku. Opisuje proceduro, v kateri so nove kapacitete zasedene glede na potrebe in sproščene, ko le-te niso več potrebne. V kontekstu računalništva v oblaku, elastičnost pomeni, da v primeru preobremenjenosti sistema, do katerega lahko pride na podlagi različnih težav (npr. nenadno povečanje števila uporabnikov), izenačevalec obremenitev (angl. Load Balancer) samodejno integrira nova vozlišča (angl. Nodes) iz lokalnega ali tudi drugega oblaka (v primeru, da pride do zapolnitve kapacitet lokalnega, privatnega oblaka). Velika prednost elastičnosti je ta, da je visoko-skalabilen sistem dostopen brez potrebne interakcije s človekom. Aplikacija lahko samodejno skalira v odvisnosti od obremenitev in tako bistveno zmanjša stroške ter izboljša izkoriščenost infrastrukturnih virov. Problem pri skalabilnosti v splošnem predstavlja to, da breme aplikacije po navadi ni enakomerno porazdeljeno preko daljšega časovnega obdobja in da so kapacitete lokalnega privatnega oblaka običajno omejene ter je potrebno aplikacije/podatke migrirati tudi na zunanji, javni oblak. Med božičnim časom so na primer, zahteve po Amazonovi spletni trgovini veliko večje kot običajno. V takšnih primerih se mora aplikacija prilagoditi obremenitvam, tako da se v primeru preobremenjenosti vozlišča dodajo oziroma v nasprotnem primeru samodejno odstranijo, po potrebi tudi posegajo po kapacitetah javnega oblaka.

Samopostrežba

Pristop računalništva v oblaku predstavlja v IT-ju evolucijo gostovanja (angl. Hosting) in podatkovnih centrov, ki omogočajo dostop do strojne ali programske opreme, omrežne tehnologije ter podpore do oddaljenih, dislociranih storitev. Inovacija oz. dodana vrednost računalniških oblakov predstavlja omogočeno upravljanje in konfiguriranje infrastrukture s strani uporabnika. Samopostrežba na zahtevo omogoča oskrbovanje, postavitev in dostop do storitev preko samopostrežnega modela. Vmesnik samopostrežbe, ki je namenjen oskrbovanju strežnikov, izenačevanju obremenitve (angl. Load Balancing), podpori podatkov in programske opreme, predstavlja osnovni nabor poslovnih zahtev. Vmesnik ali nadzorna plošča uspešnega ponudnika oblaka mora biti dostopna kadarkoli in kjerkoli, pri čemer mora zagotavljati visoko zanesljivost računalniških virov ter samega sistema. Integrirano poročanje (pogled storitev v uporabi ali izvajanju) predstavlja lastnost, ki je potrebna pri vsakodnevnem upravljanju nameščenih storitev.

Večnajemniški model

Z namenom večje podpore streženja več uporabnikov se morajo računalniški viri nahajati v bazenih. Pri tem so različni fizični in virtualni viri dinamično dodeljeni in odvzeti glede na potrebe uporabnika. Večnajemniški model daje občutek lokacijske neodvisnosti, pri čemer stranka nima nadzora oziroma vednosti o točni lokaciji ponujenih virov, vendar ima možnost specificirati lokacijo na višjem nivoju abstrakcije (npr. država, podatkovni center). Primeri virov vključujejo shrambo, procesno moč, pomnilnik, pasovno širino in virtualne stroje. Vse oblike računalniškega oblaka so predstavljene kot rešitve, ki podpirajo večnajemniški model oz. deljeno/skupno infrastrukturo. V primeru privatnih oblakov, ki so namenjeni izključno eni stranki oz. organizaciji, je fizična infrastruktura deljena med večjim številom različnih internih poslovnih enot. V nasprotju s privatnimi oblaki pa javni oblaki uporabljajo agregirane platforme, ki si jih stranke med seboj delijo, pri tem pa ni nujno, da so poslovno povezane.

Merjenje storitev

Sistemi v oblaku samodejno nadzirajo in optimizirajo porabo virov s pomočjo spremljanja porabe na določenem nivoju abstrakcije, ki je primeren za posamezen tip storitve (shramba, procesiranje, pasovna širina ali aktivni uporabniški računi). Uporabo virov je mogoče spremljati (angl. Monitor), nadzirati (angl. Control) in poročati (angl. Report) na način, ki je transparenten tako za ponudnika, kot tudi za uporabnika storitve. Organizacije plačujejo zgolj za tiste vire, ki jih dejansko porabijo, zato je komponenta, ki zagotavlja merjenje porabe virov in storitev, kritičnega pomena. Večina organizacij in ponudnikov najema programsko opremo pri zunanjih izvajalcih z razlogom izvedbe zaračunavanja ter utililizacije virov.

Konsolidacija in centralizacija

S ponujanjem skupne, adaptivne platforme lahko računalniški oblak zamenja obstoječe heterogene, pogosto neizkoriščene strežnike in shrambo s standardizirano infrastrukturo za izvajanje in skaliranje internih aplikacij. Ko so ti enotni viri oblaka povezani v poslovno omrežje z VPN (Virtual Private Network) ali katero drugo tehnologijo, lahko podjetja koristijo zmožnosti oblaka kot razširitev internih virov IT ali kot rešitev za kontinuirano okrevanje po incidentu [83].

Migracija virtualnih strojev

Migracija virtualnih strojev predstavlja dobro vpeljan pristop prenosa računalniških virov v okolju LAN (Local Area Network). Ker migracija virtualnih strojev nima bistvenega vpliva na aplikacijski nivo, je še posebej atraktivna tudi v okoljih oblakov. Mehanizmi povezljivosti pri omrežjih LAN omogočajo razširitev uporabe ter mehanizmov v okolja WAN (Wide Area Network). Zaradi vpliva odzivnosti oziroma latence v okolju WAN, moramo biti pri zagotavljanju učinkovitosti in zmogljivosti mehanizma migracije virtualnih strojev precej pazljivi.

Replikacija podatkov

Shramba je pomembna komponenta računalniških virov, ki jih uporabljamo v oblaku. Kadarkoli uporabljamo vire v oblaku, je pomembno, da celovito preučimo porabo tako računske moči kot tudi shrambe. Za vzpostavitev učinkovitosti aplikacij, latence in pasovne širine je pomembno, da so računalniški viri in podatki na skupni lokaciji. Prenos podatkov, znotraj ali med podatkovnimi centri, lahko zahteva precejšnjo količino časa. Ta čas pogosto odraža celotno trajanje migracije in je običajno daljši od same migracije virtualnega stroja. Pri prenosu podatkov je pomembno, da potrebo po migraciji predvidimo, saj se podatki replicirajo, medtem ko se aplikacija še vedno izvaja.

Replikacijo podatkov lahko v osnovi razdelimo na sinhrono in asinhrono. Pri sinhroni replikaciji se vsak blok podatkov, ki ga shranimo na lokalni sistem, hkrati shrani še na oddaljeni lokaciji in šele nato se shranjevanje na lokalnem sistemu zaključi. Pri asinhroni replikaciji pa podatkov ne shranjujemo hkrati na lokalni sistem in oddaljeno lokacijo. Shranjevanje se v tem primeru zaključi že ob uspešnem shranjevanju na lokalni sistem, kar lahko privede do nekonsistence med lokalnimi in oddaljenimi podatki. Zaradi dostopnega časa in drugih omejitev sinhrone replikacije se pogosto uporablja kombinacija asinhrone in sinhrone replikacije [85].

2.2.3 Storitveni modeli

Glede na definicijo NIST, lahko storitvene modele računalništva v oblaku kategoriziramo v tri skupine [20]:

- *Programska oprema kot storitev (SaaS)* – Končni uporabnik uporablja aplikacije, ki se izvajajo v oblaku. Aplikacije so dostopne iz številnih odjemalskih naprav, kot sta šibki odjemalec (npr. spletni brskalnik) ali programski vmesnik.

Uporabnik ne upravlja in nadzira spodaj ležeče infrastrukture oblaka, vključno z omrežjem, strežniki, operacijskimi sistemi in shrambo. Ravno tako ne nadzira in upravlja individualnih aplikacij, z možno izjemo nastavljanja uporabniško-specifičnih konfiguracij aplikacije.

- *Platforma kot storitev (PaaS)* – Stranke postavijo lastno-razvite aplikacije na infrastrukturo oblaka ali pridobljene aplikacije, ki so ustvarjene z uporabo programskih jezikov, knjižnic, storitev in orodij podprtih s strani ponudnika. Uporabnik ne upravlja in nadzira spodaj ležeče infrastrukture oblaka, vključno z omrežjem, strežniki, operacijskimi sistemi in shrambo, ampak ima nadzor nad postavljenimi aplikacijami in konfiguracijskimi nastavitvami za gostujoče okolje aplikacije.

- *Infrastruktura kot storitev (IaaS)* – Uporabnik oskrbuje procesne, podatkovne in omrežne kapacitete ter druge računalniške vire. Uporabnik lahko postavi in zaganja poljubno programsko opremo, ki vključuje operacijski sistem in aplikacije. Le-ta ne upravlja in nadzira infrastrukture oblaka, temveč ima nadzor nad operacijskim sistemom, shrambo, postavljenimi aplikacijami in ima običajno omejen nadzor nad omrežnimi komponentami.

2.2.4 Namestitveni modeli

Topologijo računalništva v oblaku sestavljajo štirje namestitvenih modeli: privatni oblak, oblak skupnosti, javni oblak ter hibridni oblak [20]. Po analizah Gartnerja in ostalih analitskih družb, so ravno privatni oblaki najbolj zanimivi za večja podjetja in organizacije, saj predstavljajo najmanjše tveganje, omogočajo izhodno strategijo v primeru, da se uporaba oblaka ne bi obnesla, obenem pa omogočajo zelo podobne prihranke, kakor javni oblaki. Nasprotno velja za srednja/manjša podjetja in organizacije, kjer potrebe po strežniških zmogljivostih niso tako velike, temveč jim oblak omogoča hitrejše lansiranje produkta na tržišče ter eliminacijo kapitalnih in operativnih stroškov potrebnih za nakup in vzdrževanje fizične infrastrukture.

Privatni oblak

Privatni oblaki so zgrajeni izključno za eno organizacijo. Organizacije se za to možnost običajno odločijo zaradi nezaupanja v podatkovno varnost ter strahu pred izgubo nadzora nad lastnimi podatki. Obstajata dva tipa javnih oblakov: notranji in zunanji privatni oblak. Notranji privatni oblaki so nameščeni znotraj podatkovnega centra

neke organizacije. Ta model predstavlja bolj standardiziran proces in zaščito, vendar je na drugi strani omejen z velikostjo in skalabilnostjo. Organizacije IT so v primeru uporabe internega privatnega oblaka izpostavljene relativno visokim kapitalnim ter operativnim stroškom. Notranji tip privatnega oblaka je najbolj primeren za aplikacije, ki zahtevajo popolni nadzor infrastrukture in varnosti. Najbolj odmevni primeri takšnega tipa oblakov so: OpenStack, Eucalyptus, OpenNebula ter Microsoft Private Cloud te VMware vCloud. Zunanji privatni oblak je na drugi strani nameščen pri ponudniku javnega oblaka, izven organizacije. V tem primeru je ponudnik tisti, ki mora zagotoviti popolno zasebnost okolja znotraj oblaka. Ta model je primeren za organizacije, ki ne želijo deliti fizičnih virov v javnih oblakih. Najbolj razširjen in sprejet primer takšnega tipa oblakov je Amazon Virtual Private Cloud (Amazon VPC) [86].

Oblak skupnosti

V primeru oblak skupnosti (angl. Community Cloud) je infrastruktura oblaka namenjena izključni uporabi specifične skupnosti uporabnikov iz organizacij, ki imajo podobne oz. skupne zahteve (npr. skupno poslanstvo, zahteve po varnosti, politike ter skladnosti s predpisi). Oblak skupnosti si lahko lasti ter z njim operira ena ali več organizacij znotraj skupnosti, tretja oseba, ali kombinacija le-teh. Lahko obstaja na lokaciji (angl. On Premises) ali izven lokacije (angl. Off Premises).

Javni oblak

Javni oblaki so v lasti tretje organizacije, ki posamezni stranki ponuja nizko cenovni model "plačaj glede na uporabo" (angl. Pay-As-You-Go). Znotraj javnega oblaka si vsi uporabniki delijo skupne računalniške vire, ki so upravljani in nadzirani s strani ponudnika javnega oblaka. Največja prednost javnega oblaka je tipična ponudba enormne kapacitete, ki omogočajo veliko elastičnost zmogljivosti in skalabilnosti. Kljub temu običajno ne omogočajo visoke stopnje nadzora. Javni oblak si lahko lastijo in z njim operirajo podjetja, akademske, raziskovalne ali vladne organizacije. Nahaja se na lokaciji ponudnika oblaka, ki je običajno geografsko razpršena na globalni ravni.

Hibridni oblak

Zraven oblakov skupnosti ter javnih in privatnih oblakov poznamo še hibridne oblake, kjer določene dele informacijskega sistema namestimo na javne, ostale dele pa na

privatne oblake ali oblake skupnosti. Hibridni oblaki torej predstavljajo kompozicijo privatnega modela, modela skupnosti ter javnega namestitvenega modela, ki ostanejo unikatne entitete, vendar jih povezujejo standardizirane, lastniške tehnologije, ki omogočajo portabilnost podatkov in aplikacij (npr. "cloudbursting" izenačevalca obremenitve med različnimi oblaki) [20, 68]. V omenjenem primeru lahko organizacije uporabljajo javne oblake z namenom povečanja fleksibilnosti svojih podatkovnih centrov. Uporaba hibridnega oblaka ponuja možnost skaliranja na zahtevo in s tem bistveno zmanjša delovne obremenitve aplikacije.

2.2.5 Izhodna strategija

Večina ponudnikov IaaS in PaaS uporablja unikatne in lastniške uporabniške vmesnike, API-je in podatkovne baze. Uporaba takšnih storitev v polni meri od uporabnikov zahteva, da programirajo v skladu s predpisanimi specifikacijami ponudnika. Pri zamenjavi ponudnikov, ki so potrebni zaradi različnih razlogov (npr. nezadovoljstvo, ponudnik preneha poslovati itn.), lahko pride do številnih težav, predvsem zaradi nestandardiziranih formatov (npr. prenos statičnih podatkov iz Amazon S3 na Azure Storage Blobs). Rešitev lahko predstavlja (1) uporaba ogrodij "multi-cloud", ki zagotavljajo neodvisnost od ponudnika storitev oblaka, na primer jclouds [87], Dasein Cloud API [88] in Deltacloud [89], ali (2) standardizacija storitev oblaka.

2.2.6 Programski modeli

Skozi zgodovino so se razvijali in uveljavljali številni programski modeli in tehnike razvoja programske opreme in aplikacij. V nadaljevanju kronološko opišemo nekatere najpomembnejše:

- *Odjemalec/Strežnik* (angl. Client/Server) – sistem je deljen na aplikacije, kjer odjemalec pošilja zahteve strežniku. V večini primerov je strežnik podatkovna baza z aplikacijsko logiko, ki je predstavljena kot bazna procedura.
- *Komponentna arhitektura* (angl. Component-Based Architecture) – dekompozicija aplikacijske arhitekture v ponovno uporabljive funkcionalne in logične komponente, ki izpostavljajo dobro definirane komunikacijske vmesnike.
- *Domensko vodeno načrtovanje* (angl. Domain Driven Design – DDD) – objektno orientiran arhitekturni stil, osredotočen na modeliranje poslovnih domen

in definiranju poslovnih objektov, ki temeljijo na entitetah znotraj poslovne domene.

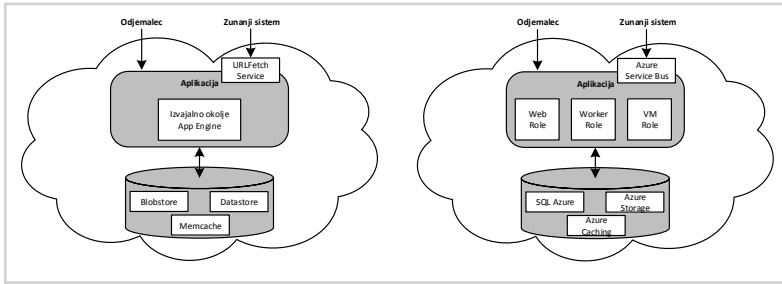
- *Nivojska arhitektura* (angl. Layered Architecture) – osredotoča se na grupiranje sorodnih funkcionalnosti aplikacije v različne nivoje, katerih funkcionalnosti so povezane s skupno vlogo ali odgovornostjo.
- *Sporočilno vodilo* (angl. Message Bus) – arhitekturni stil, ki predpisuje uporabo programske opreme za pošiljanje in sprejemanje sporočil, z uporabo enega ali več komunikacijskih kanalov. S tem se vzpostavi interakcija različnih aplikacij, kjer se le-te ne zavedajo podrobnosti drugih.
- *Večslojna/3-slojna arhitektura* (angl. N-Tier/3-Tier Architecture) – združuje funkcionalnosti v ločene segmente na podoben način kot nivojski arhitekturni stil. Pri večslojni arhitekturi vsak segment predstavlja logičen nivo, ki je lociran na drugem fizičnem računalniku.
- *Storitveno usmerjena arhitektura (SOA)* – arhitektura aplikacij, ki izpostavljajo in koristijo funkcionalnosti v obliki storitev z uporabo pogodb in sporočil.

Model računalništva v oblaku

Računalništvo v oblaku predstavlja naslednji evolucijski korak in prinaša številne prednosti, kot so elastičnost, avtomatizacija, visoka stopnja razpoložljivosti in zanesljivosti ter agilnost. Ravno tako vpeljuje nov način dostave storitev, kot so relacijska podatkovna baza v oblaku, baza NoSQL, samodejno skaliranje, varnostne skupine itn. Za uspešen izkoristek vseh omenjenih prednosti, morajo biti aplikacije za oblak načrtovane na način, ki omogoča sledenje specifičnemu aplikacijskemu modelu. Različni ponudniki storitev oblaka imajo lahko različne zahteve in s tem drugačne modele aplikacij. Primerjava programskega modela tehnologije Windows Azure ter Google App Engine je prikazana na sliki 2.2.

2.3 Zagotavljanje SLA-jev v oblaku

Računalništvo v oblaku uporabnikom ponuja uporabo spletnih storitev v obliki poslovnega modela, kjer je plačilo stroškov vezano na dejansko porabo virov. Računalništvo v oblaku ne zagotavlja nujne dolgoročne pogodbene vezave s ponudniki storitev. Z



Slika 2.2

Primerjava programskih modelov Windows Azure in Google App Engine.

namenom optimizacije uporabe virov in ponujanju fleksibilnih rešitev za uporabnike, predstavljeno področje temelji na tehnologiji virtualizacije in SOA. Določena stopnja varnosti uporabnikov, ponudnikov in računalniških virov je zagotovljena z uporabo sporazuma na ravni storitev (SLA), ki opredeljuje obseg uporabe in oskrbe virov [53]. Znotraj računalništva v oblaku je SLA nujen za prenos infrastrukture uporabnikov v podatkovne centre računalniškega oblaka. S tem se zagotavlja zanesljivost ponujenih virov in omogoča željen nivo produktivnosti. Ponudniki računalniškega oblaka uporabljajo SLA za opredelitev zaupanja in kakovosti storitev (angl. Quality of Service – QoS), ki jo ponujajo svojim uporabnikom, kakor tudi dogovorjeno ogrodje (angl. Agreed Framework) za stroške in ceno. Metrike SLA v obstoječih tehnologijah niso primerne za računalniški oblak, saj je narava in tip računalniških virov, ki so ponujeni in dostavljeni končnim uporabnikom, popolnoma drugačna. Iz tega razloga se od novih modelov SLA zahteva, da ponujajo fleksibilne metode za pogajanje in sklepanje elektronskih pogodb (angl. Electronic Contracts) med uporabnikom in ponudnikom storitev računalništva v oblaku [90].

Dogovor o nivoju storitve predstavlja dokument, ki definira zvezo med dvema strankama: ponudnikom in uporabnikom. Dogovor predstavlja pomembno dokumentacijo obeh strank in mora zagotoviti naslednje:

- identificirati in definirati potrebe uporabnika,
- ponuditi ogrodje za razumevanje,
- poenostaviti kompleksne težave,
- odstraniti konfliktna področja,

- vzpostaviti dialog v primerih konflikta in nesoglasij,
- eliminirati nerealna pričakovanja.

2.3.1 Predstavitev vsebine dokumenta SLA

V tem poglavju bomo na kratko predstavili vsebino dokumenta SLA, ki vključuje definicijo storitev, upravljanje zmogljivosti, upravljanje problemov, odgovornosti in dolžnosti strank, garancije in popravila, varnost, okrevanje po incidentu in poslovno kontinuiteto ter preklic.

Definicija storitev

Ta del predstavlja najbolj kritično sekcijo dogovora, saj opisuje storitve in način, kako morajo biti te storitve dostavljene. Standardne storitve so običajno ločene od kostumiziranih storitev, vendar ta distinkcija ni kritična. Informacije o storitvi morajo biti točne in morajo vsebovati podrobne specifikacije o tem, kar je dostavljeno.

Upravljanje zmogljivosti

Ključni del SLA se ukvarja s spremljanjem in merjenjem zmogljivosti na nivoju storitve. V osnovi, mora vsaka storitev dopuščati merjenje, analizo rezultatov in poročanje. Preizkusni rezultati (angl. Benchmarks), cilji in metrike morajo biti specificirani znotraj SLA dogovora. Nivo zmogljivosti storitve mora biti preverjen v rednih časovnih intervalih s strani vsaj dveh strank.

Upravljanje problemov

Bistvo upravljanja problemov je v minimizaciji vpliva neprijetnih situacij (incidentov) in ostalih težav. To običajno pomeni, da mora obstajati zadosten proces, ki bo razreševal nenačrtovane incidente in preventivno zmanjšal pojav neprijetnih situacij, ki predstavljajo vzrok pojava nenapovedanih situacij.

Odgovornosti in dolžnosti strank

Zelo pomembno je, da se stranka pri dostavi storitve zaveda svojih obveznosti in odgovornosti. SLA definira zvezo, ki predstavlja dvosmerno entiteto. Tipično je stranka tista, ki mora urediti dostop, prostor in vire za zaposlene pri dobavitelju.

Garancije in popravila

Vsebina dokumenta SLA, ki služi garanciji in popravilu običajno pokriva naslednja ključna poglavja: QoS, zavarovanja (angl. Indemnities), napade od zunaj, popravila ob napakah in izključitve.

Varnost

Varnost je še posebej kritična funkcija vsakega dokumenta SLA. Stranki mora biti ponujen nadzorovan fizični in logični dostop do svojih virov in informacij. Ponudnik mora iz tega razloga spoštovati in upoštevati varnostne politike uporabnikov in procedure.

Okrevanje po incidentu in poslovna kontinuiteta

Okrevanje po incidentu (angl. Disaster Recovery) in poslovna kontinuiteta (angl. Business Continuity) sta zelo pomembna aspekta računalniških oblakov in se posledično moreta upoštevati pri izdelavi dokumenta SLA. Okrevanje po incidentu je običajno obravnavano v sekciji varnosti, vendar je prav tako pogosto vključeno v sekcijo upravljanja problemov. Obe področji tipično pravita, da mora biti prisoten mehanizem oskrbovanja za načrtovanje okrevanja po incidentu in poslovne kontinuitete, da bi zagotovili brezhibno in trajno delovanje dostavljenih storitev.

Preklic

Ta sekcija dogovora SLA tipično pokriva naslednja področja:

- preklic ob koncu začetnega obdobja,
- preklic zaradi primernosti,
- preklic iz razloga,
- plačilo ob preklicu.

2.3.2 Ogrodja dogovora na ravni storitve - SLA

Ključne specifikacije, ki so oblikovane za opisovanje sintakse strukture SLA so:

- Dogovor o spletnih storitvah (angl. Web Service Agreement – WS-Agreement [91]).

- Dogovor na ravni spletnih storitev (angl. Web Service Level Agreement – WSLA [53]).

Dogovor o spletnih storitvah – WS-Agreement

Skupnost Open Grid Forum (OGF) je z namenom, da bi ustvarila uradno pogodbo med uporabnikom storitve in ponudnikom storitev predlagala dogovor o spletnih storitvah. WS – Agreement specificira zagotovila, obveznosti in kazenske globe v primerih kršitev pogodbe. Funkcionalne zahteve in ostale specifikacije storitev so ravno tako lahko vključene v SLA. Obstajajo tri ključne sekcije dogovora o spletnih storitvah [91]:

- *Ime* (angl. Name) – predstavlja enoličen identifikator. Imena storitev so opcijsko vključena v imensko sekcijo.
- *Kontekst* (angl. Context) – vsebuje informacije o uporabniku storitve in ponudniku le-te, domeni storitve in ostale specifikacije storitev.
- *Pogoji* (angl. Terms) – vsebuje dodatne pogoje storitev in zagotovila, ki so opisana podrobneje.

Obstoječi SLA-ji so bili razviti za uporabo splošnih storitev (angl. General Services). Uporabniki storitev se znotraj računalništva v oblaku srečujejo s pomanjkanjem specifičnih storitev za SLA, s katerimi bi lahko predstavili ključne parametre virtualiziranega okolja oblaka. Omenjene storitve bi morale biti dinamično integrirane s poslovnimi pravili uporabnikov oblaka.

Dogovor na ravni spletnih storitev – WSLA

Zaradi vedno večje težnje po delegiranju opravil, aplikacij in podatkov k ponudnikom računalništva v oblaku, s katerimi se srečujejo številni uporabniki računalništva v oblaku, je pomembno, da se upošteva dogovor na ravni storitve med uporabnikom/odjemalcem in ponudnikom. Dinamično okolje oblaka pri podpori SLA zahteva neprestano spremljanje atributov kakovosti storitve (QoS). Dodatno je potrebno upoštevati še ostale faktorje, kot je npr. zaupanje, ki je danes eden ključnih ovir, zakaj podjetja ne želijo migrirati svojih podatkov in aplikacij v oblak. V nadaljevanju bomo definirali strukturo SLA, ki uporablja ogrodje WSLA in temelji na storitveno usmerjeni arhitekturi.

Računalništvo v oblaku predstavlja nov trend računalništva, kjer so hitro razpoložljivi računalniški viri izpostavljeni in dostopni v obliki storitve. Ti računalniški viri so običajno ponujeni v obliki poslovnega modela "plačaj glede na uporabo". Medtem ko se uporabniki nagibajo k prisvojitvi takšne storitveno usmerjene arhitekture, postaja kakovost in zanesljivost storitev zelo pomemben vidik. Kakorkoli, potrebe po storitvah s strani uporabnikov zelo variirajo. Zelo težko je zadovoljiti pričakovanja vseh uporabnikov s perspektive ponudnika storitev. Potrebno je zagotoviti ravnovesje v procesu pogajanja. Na koncu procesa pogajanja se morata ponudnik in stranka zavezati k dogovoru. Po pojmovanjih koncepta SOA, takšen dogovor imenujemo SLA. SLA služi kot temelj pričakovanega nivoja storitve med uporabnikom in ponudnikom. Atributi QoS (kot sta na primer odzivni čas in prenos podatkov), ki so običajno del pogodbe SLA, se konstantno spreminjajo in da bi vpeljali dogovor, morajo biti ti parametri podrobno spremljani.

Specifikacija WSLA se sestoji iz nabora konceptov in jezika XML. Oblikovana je z namenom zajeti dogovore SLA na formalen način in za opisovanje storitev v treh kategorijah:

- *Stranke* (angl. Parties) – v tej kategoriji so opisane informacije o uporabniku storitev, ponudniku storitev ter agentih.

- *Parametri SLA* (angl. SLA Parameters) – v tej kategoriji so predstavljeni ključni parametri, ki so merljivi, in to v dveh tipih metrik. Prvi tip metrik so metrike računalniških virov (angl. Resource Metrics), ki jih uporabljamo za opis virov ponudnika storitev v obliki vrstičnih informacij. Drugi tip metrik so kompozitne metrike (angl. Composite Metrics), ki so uporabljene za predstavitev izračunov kombinacij informacij o virih ponudnika storitev.

- *Cilj na ravni storitev* (angl. Service Level Objective – SLO) – ta kategorija je uporabljena za specificiranje dolžnosti in ukrepov v primeru neupoštevanja zagotovil storitev v pogodbi SLA s strani ponudnika ali uporabnika storitev računalniškega oblaka. SLO predstavlja ključen del dokumenta SLA, ki je sklenjen med ponudnikom storitve in stranko. SLO-ji predstavljajo dogovorjeno sredstvo za merjenje zmogljivosti/učinkovitosti ponudnika storitev ter so namenjeni izogibanju sporov med dvema strankama. Predstavlja specifične merljive karakteristike vsakega dokumenta SLA, kot so razpoložljivost (angl. Availability), pretok (angl.

Tabela 2.1

Primer ciljev na ravni storitev – SLO.

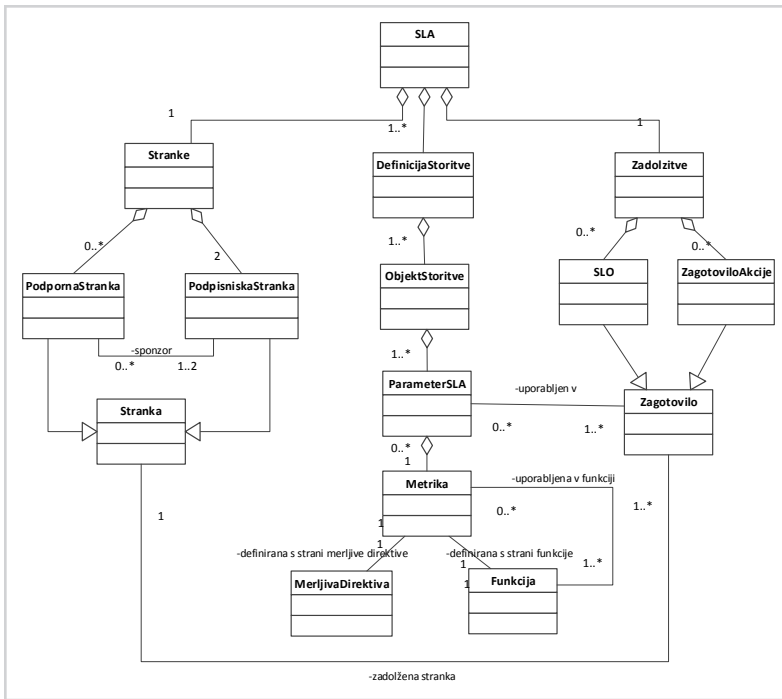
Parametri SLA	*Operator*	*Vrednost*
• Vhodna pasovna širina	>	10 Mbit/s
• Izhodna pasovna širina	>	12 Mbit/s
• Shramba	>	1024 GB
• Razpoložljivost	>=	99%

Throughput), pasovna širina (angl. Throughput), odzivni čas (angl. Response Time) ali kakovost (angl. Quality), kot prikazuje tabela 2.1.

Slika 2.3 prikazuje vse ključne komponente strukture WSLA. Struktura WSLA je upodobljena z uporabo diagramske tehnike UML, kjer WSLA vsebuje naslednja večja področja:

- *Stranke* (angl. Parties) – področje sestavljata dva tipa strank: podporne stranke (angl. Supporting Parties) in podpisniške stranke (angl. Signatory Parties). Podpisniške stranke predstavljata ponudnik in uporabnik storitev. Podporne stranke pa predstavljajo tretje stranke, ki pridejo v poštev takrat, ko se podpisniške stranke odločijo delegirati določena opravila (npr. merljivi parametri SLA).
- *Definicije storitev* (angl. Service Definition) – definicija storitev vsebuje opis vmesnika ponudnikov storitev. Storitve so predstavljene kot objekti storitev (angl. Service Objects). Vsak objekt storitev je povezan z enim ali več parametri SLA.
- *Zadolžitve* (angl. Obligations) – zadolžitve predstavljajo pogoje in zagotovila o dejanjih (angl. Action Guarantees) [53].

V nadaljevanju predstavljamo strukturo WSLA, ki je prilagojena za okolje računalništva v oblaku. Predpostavimo lahko, da sta znotraj arhitekture računalništva v oblaku, uporabnik oblaka in ponudnik že sodelovala v procesu pogajanja in posledično imata dogovorjen nabor parametrov. Po vzpostavitvi dokumenta SLA je potrebno izvesti še njegovo namestitev (angl. Deploy). Terminologija namestitve SLA (angl. SLA Deployment) je definirana kot proces validacije in distribucije SLA-jev k vpletenim strankam.



Slika 2.3

Osnovni koncepti WSLA [53].

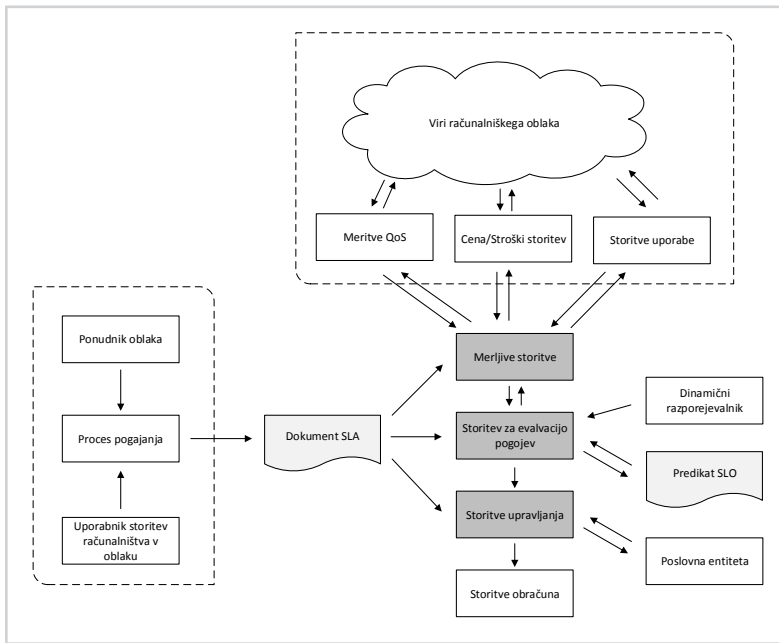
Pri namestitvi SLA obstaja možnost, da si ponudnik in uporabnik zaradi varnostnih razlogov ne bi želela deliti popolnega dokumenta SLA s podpornimi strankami. Skladno s tem v nadaljevanju opišemo tri običajne storitve WSLA in nekatere potrebne adaptacije v kontekstu oblaka:

- *Merljive storitve* (angl. Measurement Services) – storitve so odgovorne za merjenje časa izvajanja parametrov za vire ponudnika oblaka. Storitveni parametri, kot sta odzivni čas in prepustnost, se konstantno spreminjata zaradi variabilnosti zahtev po storitvah s strani končnega uporabnika. V kontekstu oblaka je uporaba in cena parametrov dinamična. Vzrok je v poslovnem modelu "plačaj glede na uporabo" in elastični naravi oblaka. Pri teh storitvah sta identificirani dve dodatni storitvi: poraba in cena podatkov. Le-ti je potrebno vključiti k naboru merljivih storitev v kontekstu računalništva v oblaku.

- *Storitev za evalvacijo pogojev* (angl. Condition Evaluation Service) – storitev je zadolžena za pridobivanje rezultatov iz merljivih storitev in evalvacijo ciljev na ravni storitev. V primeru kršitve je obveščena upravljavska storitev. Zaradi dinamične narave oblaka je potrebno evalvirati pogoje bolj pogosto kot pri ogrodjih tradicionalnih storitev. Pri tem se kompleksnosti pogojev pogosto ne namenja veliko pozornosti. Skladno s tem morajo biti za hitrejšo zagotovitev ciklov evalvacije takšni pogoji v kontekstu oblaka preprostejši. Storitev za evalvacijo pogojev ima dodan dinamični razporejevalnik (angl. Dynamic Scheduler), ki je odvisen od metrik, kot je na primer stopnja transakcije. Uporaba omenjenega razporejevalnika zagotavlja, da se preverjanje visokih obremenitev izvede kontinuirano, saj je zelo verjetno, da bo prišlo do kršitve ravno med temi tranzicijami.

- *Storitev upravljanja* (angl. Management Service) – storitev je zadolžena za izvedbo ukrepov, do katerih pride zaradi kršitev SLO. Ker oblak predstavlja računalniške vire v obliki storitve, bo storitev upravljanja primarno skrbela za finančne globe, ki so podobne tistim iz industrijske prakse [53].

Primarna slabost teh pristopov (tj. WS-Agreement ali WSLA) je v tem, da ne ponujajo dinamičnega pogajanja (angl. Dynamic Negotiation). Pri tem različni tipi uporabnikov oblaka potrebujejo različne strukture implementacij SLA, da bi integrirali svoja poslovna pravila z zagotovili, ki so predstavljena v ciljnem SLA. Osnovna struktura dogovora na ravni storitev med uporabnikom in ponudnikom storitev, ki rešuje



Slika 2.4

Arhitektura specifikacije WSLA [53].

predhodno omenjene težave (poglavje 2.3.2), je opisana v nadaljevanju dokumenta. Uporabljeni sta dve bistveni kategoriji metrik SLA (slika 2.4):

- *Zmogljivostne metrike* (angl. Performance Metrics) – metrike prikazujejo rezultate meritev parametrov zmogljivosti (angl. Performance Parameters) znotraj podatkovnih centrov računalniškega oblaka, kot sta odzivnost (angl. Response Time) in kapaciteta CPU (angl. CPU Capacity).
- *Poslovne metrike* (angl. Business Metrics) – ključne meritve poslovnega aspekta, ki jih predstavljajo poslovne metrike, vključujejo poslovne atribute, kot so npr. cena posameznih storitev in metode zaračunavanja [53].

2.4 Migracija v oblak

Proces, pri katerem načrtovalci aplikacij izvedejo prenos in ponovno postavitev aplikacije na novo platformo ali infrastrukturo, se imenuje migracija. V primeru računalništva v oblaku je lahko aplikacija prenesena iz obstoječega podatkovnega centra v poljuben ciljni oblak. Ciljno infrastrukturo lahko predstavlja javni, privatni ali hibridni oblak. Slednji transparentno združuje funkcionalne zmogljivosti javnega in privatnega oblaka. Pri migraciji aplikacij lahko uporabimo tudi različne tipe oblaka, kot sta na primer IaaS ter PaaS. Pri identifikaciji aplikacij, ki jih želimo prenesti v oblak, je najprej potrebno preučiti in razumeti poslovne in tehnične faktorje migracije. Zmanjšanje stroškov in poslovna agilnost predstavljata dva tipična poslovna faktorja za migracijo obstoječih aplikacij v oblak. Računalništvo v oblaku lahko zaradi povečane utilizacije ponudi bistvene prihranke. Povečana utilizacija je posledica večnajemniškega modela (angl. Multitenancy), standardizacije ter avtomatizacije storitev oblaka. Iz operativnega vidika so upravljanje, zmogljivost ter skalabilnost tipični vzroki, zakaj podjetja razmišljajo o prisvojitvi oblaka.

2.4.1 Tipi migracije v oblak

V današnjem okolju IT številni ponudniki storitev ponujajo oskrbovanje, upravljanje in skaliranje storitev večjemu številu končnih uporabnikov. Hkrati ponujajo zmogljivosti, ki so zasnovane na nivoju IaaS, pri čemer organizacije poslujejo s ponudniki storitev na podlagi poslovnega modela "plačaj glede na uporabo". Številna podjetja in organizacije prav tako vzpostavljajo lastne privatne oblake, kjer se vpeljejo infrastrukturne storitve IT, ki jih upravlja organizacija sama. Dodatno infrastrutkurne storitve

podpirajo tudi koncepte, kot so samopostrežba, poslovni model v obliki storitve, oskrbovanje na zahtevo ter občutek neskončnosti virov.

Ne glede na storitveni model (privatni ali javni oblak) morajo nato organizacije najprej identificirati, katere aplikacije je smiselno prenesti v oblak in kako izvesti samo migracijo. Aplikacije so lahko prenesene iz obstoječih podatkovnih centrov v ciljni oblak, ki je lahko javni, privatni ali hibridni. Ko je določena aplikacija enkrat identificirana kot kandidat za migracijo v oblak, glede na poslovne in tehnične faktorje, je pomembno ugotoviti, za kateri tip računalništva v oblaku (PaaS ali IaaS) je aplikacija najbolj primerna [92].

Platforma kot storitev

Ena izmed možnosti za migracijo poslovnih aplikacij, ki temeljijo na standardnih aplikacijskih strežnikih, kot so Javanski aplikacijski strežniki ali strežniki Microsoft SQL Server, predstavlja platforma kot storitev. Pri tem modelu ponudnik storitev upravlja aplikacijsko platformo in ponuja dostop do aplikacijskih storitev, kot je na primer podatkovna baza SQL. Aplikacijska platforma je lahko skupna več aplikacijam, kjer vsaka aplikacija pripada drugi stranki. Način, kako je aplikacijska platforma povezana s fizično infrastrukturo, je običajno nadziran s strani ponudnika računalništva v oblaku. Odločitveni faktorji pri migraciji aplikacije so odvisni od tipa in verzije aplikacijskega strežnika, ki ga je aplikacija, ki jo prenašamo, predhodno uporabljala. Nekatera okolja PaaS ne podpirajo vseh funkcionalnosti aplikacijskih strežnikov in posledično zahtevajo spremembe aplikacij. Kriteriji, ki jih je potrebno upoštevati pri postavitvi PaaS, so sledeči [92, 93]:

- *Dogovor na ravni storitve (SLA)* – ponudnik storitev PaaS mora ponuditi SLA za razpoložljivost in zmogljivost aplikacijske platforme. Ponudnik mora ravno tako definirati jasno politiko in smernice za vzdrževanje in upravljanje verzij (angl. Version Management) za platformo ter politike za kompatibilnost verzij API-jev med platformo in aplikacijo.

- *Portabilnost podatkov* – podatki aplikacije so običajno shranjeni v podatkovni bazi, ki jo priskrbi ponudnik storitev oblaka. Uporabnik mora imeti možnost izvoziti podatke v format, ki bo omogočil migracijo k drugemu ponudniku. Enako velja za statične podatke, shranjene znotraj omrežnih datotečnih sistemov.

- *Dolgoročni stroški* – finančni model PaaS nivoja mora biti primerjan z modelom v internih namestitvah infrastrukture in aplikacijske platforme, ki uporablja model IaaS za postavitev aplikacijskih strežnikov na strežnike oblaka.

- *Upravljanje uporabnikov* – aplikacija postavljena na nivoju PaaS zahteva administrativne in uporabniške račune. Za oba tipa računov se mora uporabnik zavedati, kako je upravljanje z uporabniki usklajeno z obstoječimi imeniškimi storitvami (angl. Directory Services) in procesi upravljanja z uporabniki (angl. User Management Processes).

- *Varnost* – v okolju PaaS lahko isti fizični strežnik izvaja aplikacije različnih uporabnikov. V takšnih večnajemniških okoljih (angl. Multitenant Environments) so zahtevani dodatni varnostni ukrepi, ki zagotavljajo varno izolacijo posameznih aplikacij.

- *Upravljanje s platformo* – aplikacijski strežniki ponujajo upravljavske konzole (angl. Management Consoles) in orodja za spremljanje in upravljanje aplikacij, ki se izvajajo na njih. Okolje PaaS mora uporabnikom ponujati analogen nabor orodij za upravljanje in optimiziranje zmogljivosti aplikacij.

Infrastruktura kot storitev

Pri migraciji aplikacije na model IaaS je najprej potrebno ugotoviti, ali sta strežniška strojna oprema in operacijski sistem kompatibilna s trenutno strežniško strojno opremo in operacijskim sistemom (OS). V primeru, da se aplikacija izvaja na strežnikih x86, ki so grajeni na arhitekturi x86, mora ponudnik oblaka implementirati x86 inštrukcije. Če strojna oprema ni kompatibilna, je aplikacijo potrebno ponovno prevesti in postaviti na novo platformo. V primeru, da je OS kompatibilen, bo pri migraciji aplikacije potrebno le nekaj sprememb. Večina kriterijev, ki jih upoštevamo pri migraciji aplikacij v okolje PaaS, mora biti upoštevanih tudi na nivoju IaaS [92, 93]:

- *Dogovor na ravni storitve (SLA)* – v okolju nivoja IaaS je potrebno zagotoviti SLA za razpoložljivost in zmogljivost delovanja strežnikov, omrežja in shrambe podatkov. Ravno tako morajo biti zagotovljeni dogovori o vzdrževanju in upravljanju infrastrukture in dogovori o času prekinitve (angl. Downtime).

- *Portabilnost podatkov* – aplikacija lahko uporablja strežnik podatkovne baze, ki se prav tako nahaja v oblaku. V tem primeru mora ponudnik storitev oblaka zagotoviti replikacijo in migracijo blokovne shrambe ali datotečnega sistema, ki se nahaja na strežnikih obstoječega podatkovnega centra.

- *Dolgoročni stroški* – stroški aplikacije, ki je postavljena na model IaaS, morajo biti primerljivi s stroški postavitve v podatkovnih centrih organizacij. V nekaterih primerih ima javna namestitev modela IaaS očitne prednosti zaradi dinamičnega skaliranja in zaračunavanja glede na porabo. Za aplikacije v javnem oblaku so lahko dolgoročni stroški lastništva v nekaterih primerih tudi večji od stroškov lastništva v privatnem oblaku.

- *Upravljanje uporabnikov* – v okolju IaaS so zahtevane vsaj tri različne vloge uporabnikov: strežniški administrator, aplikacijski administrator in uporabnik aplikacije.

- *Varnost* – virtualni stroji, ki pripadajo različnim strankam, so lahko implementirani na skupni fizični infrastrukturi. Preden aplikacijo prenesemo v oblak, je naprej potrebno preveriti varnostne politike za virtualno in fizično izolacijo ponudnikove infrastrukture ter skladnost z zakonodajo.

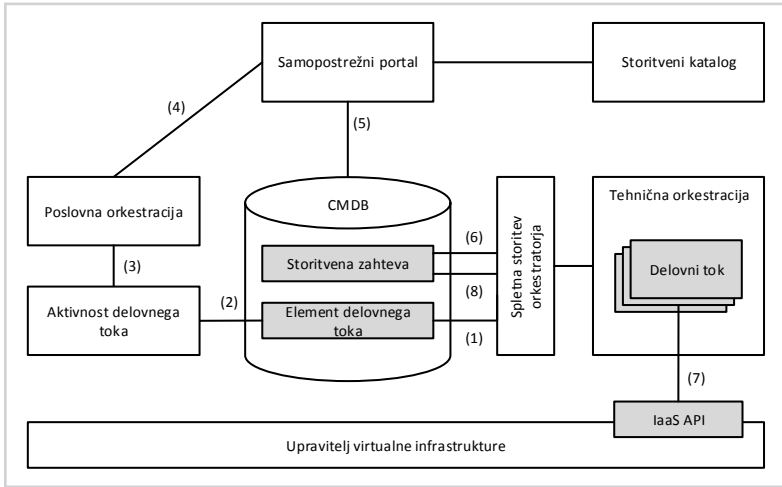
- *Skalabilnost* – aplikacije, ki so zasnovane za horizontalno skaliranje, so običajno večslojne in imajo funkcijo za izenačevanje obremenitev, ki omogoča, da so lahko aplikacije brez stanj (angl. Stateless Applications) skalirane navzgor ali navzdol. V primeru uporabe te funkcije mora ponudnik storitev oblaka ponuditi jasno politiko o tem, kako bo ta tip skaliranja funkcioniral in kako bo poskrbljeno za dodeljevanje virov med več uporabniki in aplikacijami.

2.4.2 *Prednosti in slabosti migracije v oblak*

Računalništvo v oblaku ponuja organizacijam številne prednosti. Prva večja prednost migracije v javni oblak je ta, da organizacijam ni potrebno veliko investirati v fizično in programsko infrastrukturo, potrebna je zgolj dobra internetna povezava. Ravno tako so eliminirani operativni stroški za vzdrževanje in upravljanje infrastrukture. Elastičnost oblaka, kjer se kapacitete avtomatično razširijo glede na potrebe, je ključna prednost, ki diferencira računalniški oblak od ostalih oblik gostovanja [2]. Računalništvo v oblaku tipično operira s poslovnim modelom ”plačaj glede na uporabo”, kar nam omogoča,

da dejansko najamemo toliko računalniških virov in storitev, kot jih tudi potrebujemo. Naslednja prednost migracije predstavlja možnost dostopa do aplikacij in podatkov iz katerekoli naprave, ki ima dostop do interneta. Oblak prav tako zagotavlja visoko stopnjo zanesljivosti in razpoložljivosti, saj omogoča replikacijo podatkov tudi med različnimi podatkovnimi centri in številne mehanizme za okrevanje po incidentu [93, 94]. Ena izmed pomembnih prednosti prenosljivosti aplikacij in podatkov v oblak je agilnost, ki omogoča organizacijam večjo odzivnost na spremembe pri poslovanju in posledično hitrejše lansiranje novih storitev na tržišče. Migracija aplikacij v oblak ravno tako zagotavlja neodvisnost od naprave in lokacije uporabnika, ter s tem poveča dostopnost do aplikacij in podatkov [95].

Računalništvo v oblaku podpira izvajanje storitev in aplikacij, ki so že nameščene in dostopne v enem ali več podatkovnih centrov. Aplikacije, ki najbolj ustrezajo okolju računalništva v oblaku, so običajno tiste, ki strežejo veliki količini uporabnikov, kot so na primer e-poštne aplikacije, različni tipi splošno-namenskih spletnih aplikacij, socialna omrežja, sistemi ERP (Enterprise Resource Planning) in CRM (Customer Relationship Management) [93]. Ena izmed ključnih ovir pri migraciji aplikacij v oblak je zaklepanje znotraj ponudnika. To pomeni, da ko se enkrat odločimo za določenega ponudnika storitev računalništva v oblaku, je prenos podatkov in aplikacij na druge ponudnike zelo težaven. Razlog je v tem, da različni ponudniki ponujajo lastne nestandardizirane formate podatkov in aplikacij [96]. Naslednji oviri, ki močno vplivata na odločitev migracije informacijskih rešitev v oblak, sta varnost in zasebnost podatkov. Le-ti predstavljata pri podjetjih in organizacijah daleč največjo skrb, še posebej v primeru prenosa poslovno-kritičnih podatkov v roke zunanjemu ponudniku [97]. Geografska lokacija predstavlja naslednjo večjo oviro. V primeru, da se podjetja odločijo gostovati svoje aplikacije v podatkovnih centrih na območju Združenih držav Amerike, bi v skladu s predpisi zakona U.S. Patriot Act [98], imela vlada možnost vpogleda v njihove podatke. Ponudnik mora torej omogočiti svojim uporabnikom možnost izbire, v katerih podatkovnih centrih se bodo njegove aplikacije nahajale. Obstajajo še nekateri drugi pomisleki [90, 95, 97], ki so povezani z zagotavljanjem stabilne internetne povezave, visoke stopnje razpoložljivosti ter pomanjkanjem robustnih dokumentov SLA.



Slika 2.5

Interakcijska arhitektura dvonivojske orkestracije oblaka.

2.5 Orkestracija storitev oblaka

Ključno komponento računalniškega oblaka predstavlja orkestracija storitev oblaka, saj je pomembno, da so avtomatizacija in delovni tokovi iz različnih domen (npr. sistemske in poslovne) izvedeni na koordiniran, konsistenten in zanesljiv način. Pri tem ne gre samo za tehnično/sistemsko avtomatizacijo, temveč v enaki meri vključuje visokonivojske poslovne procese/delovne tokove za upravljanje infrastrukture oblaka. Skladno s tem mora biti za brezhibno avtomatizacijo preko obeh domen (od začetka do konca) uveljavljen dvonivojski orkestracijski pristop.

Slika 2.5 prikazuje interakcijo komponent v takšni dvoslojni orkestracijski arhitekturi. V tem primeru (1) konektor orkestratorja sinhronizira podatke delovnega toka z bazo CMDB (Configuration Management Database). Integrirana baza CMDB je zadolžena za vzdrževanje artefaktov kot so delovni tokovi, oblaki, predloge VM-jev, predloge storitev, VM-ji, storitve oblaka, uporabniki itn. V naslednjem koraku (2) administrator ustvari aktivnosti delovnih tokov in preslika vhodno/izhodne parametre določenega delovnega toka v poslovne lastnosti, ki so razumljive poslovnim uporabnikom. Administrator nato vključi izbrane aktivnosti delovnih tokov v predlogo storitvenih zahtev (angl. Service Request Template) (3) da bi oblikovali avtomatizirane poslovne procese/delovne tokove (tj. poslovna orkestracija). Predloga storitvene

zahteve se kasneje registrira v storitveni katalog znotraj oblaka, s katerim je asociirana in je kot taka izpostavljena v obliki storitvene ponudbe (angl. Service Offering), ki hrani dodatne informacije, kot so pogoji uporabe (angl. Terms and Conditions) in določanje cen (angl. Pricing Rates) [99]. Storitvena ponudba se nato pojavi v samopostrežnem portalu in je na voljo uporabnikom storitev za izvedbo zahtev (4). V nadaljevanju končni uporabnik ustvari storitveno zahtevo na samopostrežnem portalu in požene poslovni delovni tok (5). Za potrebe izvedbe tehnične okrestracije se proži določena aktivnost poslovnega procesa (delovnega toka) z vsemi zahtevanimi uporabniškimi vhodnimi podatki (6). Nato aktivnost tehničnega delovnega toka kliče ustrezno operacijo API-ja IaaS (7), medtem ko avtomatizira nekatera opravila, specifična za oblak (npr. oskrbovanje vnaprej konfigurirane instance VM), nad spodaj ležečim upraviteljem virtualne infrastrukture. Nazadnje komponenta spremljanja delovnih tokov preverja status delovnega toka ves čas življenjskega cikla storitvene zahteve.

"If you live each day as if it was your last, someday you'll most certainly be right."

— Steve Jobs

3

Arhitekturni model IaaS

3.1 Uvod

Arhitekturni model IaaS sestavljata med seboj povezana taksonomija IaaS ter arhitekturno ogrodje IaaS. Računalništvo v oblaku je asociirano z novo paradigmo oskrbovanja različnih računalniških virov, običajno naslovljenih iz treh fundamentalnih aspektov: infrastrukture kot storitve (IaaS), platforme kot storitve (PaaS) ter programske opreme kot storitve (SaaS). Nedvomno nivo IaaS predstavlja eno izmed najbolj zanimivih področij arhitekturnega modela [22]. Iluzija virtualno neskončnih računalniških kapacitet, poslovni model "plačaj glede na uporabo", večnajemniški model, rapidna elastičnost ipd. so nekatere izmed relevantnih funkcij [100]. Zaradi hitre rasti računalniških oblakov v svetu IT, so se pojavile številne definicije in povzročile splošno zmedo glede te paradigme in njenih zmožnosti, kar je prelevilo oblak v pretirano splošni izraz, ki vključuje skoraj vsako rešitev, ki omogoča zunanje izvajanje (angl. Outsourcing) vseh vrst gostovanja in računalniških virov [3].

Nastajajoče rešitve računalniških oblakov izpolnjujejo najpogostejše zahteve okolij IT (npr. agilnost in nadzor stroškov) [6]. V primeru agilnosti je lahko IT uporabljena kot konkurenčno orodje preko rapidne postavitve, paralelnega procesiranja, uporabe računsko-intenzivne poslovne analitike in mobilnih interaktivnih aplikacij, ki se odzivajo na uporabniške zahteve v realnem času. V primeru nadzora stroškov, je moč sodobnih računalnikov učinkovitejše utilizirana preko visoko-skalabilnih virov strojne in programske opreme. Rezultat omenjenega pristopa je izrazita redukcija stroškov. Kljub temu, številne organizacije ne koristijo prednosti rešitev IaaS, delno tudi zaradi negotovosti in pomanjkanja informacij o njihovih zmožnosti. Iz primerjave sistemov IaaS, lahko organizacije IT bolje razumejo različne platforme oblaka in bolj smiselno izberejo najprimernejšo tehnologijo. Iz omenjenega razloga je pomembno, da uporabimo mehanizem za splošno razumevanje tehnologij IaaS, kar predstavlja osnovo za pošteno primerjavo in evalvacijo.

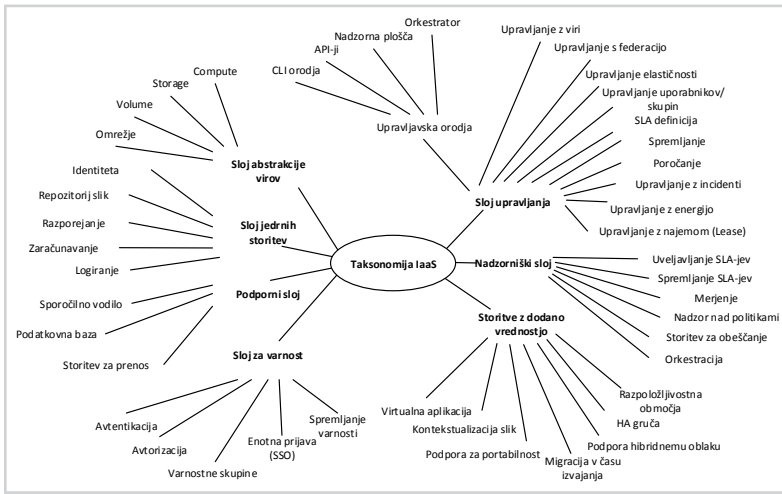
Na podlagi izkušenj, ki smo jih pridobili pri raziskovalnem delu znotraj našega laboratorija (konfiguracija in uporaba raznolikih platform IaaS, ter izdelava celovitega pregleda literature), predlagamo enotno taksonomijo ter arhitekturno ogrodje IaaS, ter dodatno izvedemo poglobljeno evalvacijo najpomembnejših platform IaaS. Enotna taksonomija je sestavljena iz sedmih slojev (vsak sloj vsebuje številne relevantne komponente): sloj jedrnih storitev, podporni sloj, storitve z dodano vrednostjo, nadzorniški sloj, sloj upravljanja, sloj varnosti ter sloj virov abstrakcije. Cilj predlagane

taksonomije je identifikacija in klasifikacija fundamentalne komponente IaaS v urejene kategorije/sloje, ter uporaba taksonomije za oblikovanje arhitekturnega ogrodja IaaS. Cilj predlaganega arhitekturnega ogrodja IaaS je (1) organizirati konceptualne sloje/komponente v arhitekturno ogrodje, (2) ponuditi skupno izhodišče za analizo, primerjavo in evalvacijo implementacijskih arhitektur IaaS, ter (3) definirati odvisnosti med posameznimi sloji in komponentami. Na podlagi taksonomije in arhitekturnega ogrodja, naredimo poglobljeno evalvacijo številnih komercialnih in odprto-kodnih platform IaaS.

3.2 Taksonomija IaaS

Taksonomija je znanost kategorizacije ali klasifikacije poljubnih stvari, ki temeljijo na vnaprej definiranem sistemu in vsebujejo nadziran besednjak s hierarhično drevesno strukturo [12]. Namen predlagane taksonomije (slika 3.1) je identificirati in klasificirati fundamentalne komponente IaaS v urejene kategorije/sloje in uporabiti taksonomijo za oblikovanje arhitekturnega ogrodja IaaS. Sloji in komponente so bili identificirani glede na (1) literarni pregled najpomembnejših komercialnih in odprto-kodnih IaaS produktov iz sveta industrije in akademije [101, 102], (2) preiskavo trenutnih in prihodnjih tehnoloških trendov paradigme IaaS [103, 104] ter (3) tehnično postavitev in testiranje individualnega sistema IaaS v sklopu številnih projektov iz realnega sveta (poglavje 3.6).

Taksonomijo sestavlja sedem ključnih slojev: sloj abstrakcije virov, sloj jedrnih storitev, podporni sloj, ki služi kot komunikacijski sloj med slojem jedrnih storitev in slojem abstrakcije virov, sloj varnosti, sloj upravljanja, nadzorniški sloj ter storitve z dodano vrednostjo. Sloj abstrakcije virov vključuje osnovne virtualizacijske vire infrastrukture računalniških oblakov (komponenta Compute, komponenta Storage, komponenta Volume ter omrežna komponenta). Sloj jedrnih storitev vključuje temeljne funkcionalnosti vsakega sistema IaaS (identiteta, repozitorij slik, razporejanje, zaračunavanje ter logiranje). Podporni sloj predstavlja ključnega posrednika med slojem jedrnih storitev ter slojem abstrakcije virov in omogoča interakcijo med ostalimi sloji arhitekture. Primer so komponente znotraj sloja jedrnih storitev, ki komunicirajo s spodaj ležečimi viri in so močno odvisne od podpornega sloja, da bi lahko uspešno izvedle svoja opravila. Te podporne komponente so sporočilno vodilo, podatkovna baza ter storitev za prenos. Sloj varnosti igra pomembno vlogo pri rešitvah IaaS, saj predstavlja varnost eno izmed večjih barier pri adaptaciji računalniških oblakov. Le-ta vključuje komponente

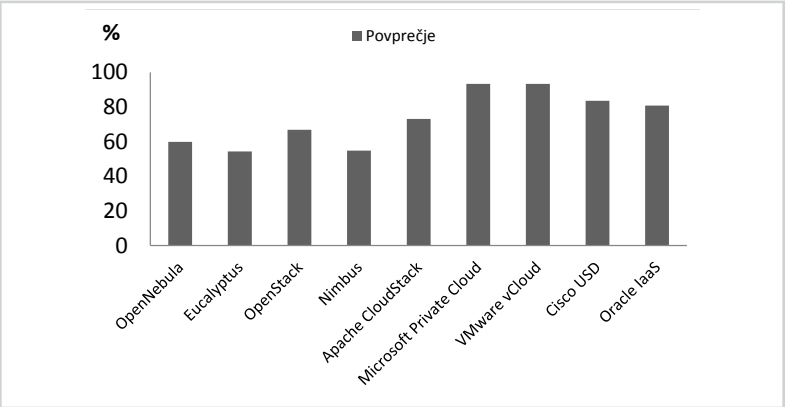


Slika 3.1

Enotna taksonomija arhitektur IaaS.

za avtentikacijo, avtorizacijo, varnostne skupine, enotno prijavo (angl. Single Sign-On – SSO) ter spremljanje varnosti. Sloj upravljanja vsebuje enajst komponent, ki so zadolžene za celovito upravljanje z infrastrukturnim računalniškim oblakom (upravljavska orodja, upravljanje s federacijo, upravljanje z viri, upravljanje avtomatizacije, upravljanje uporabnikov/skupin, definicija SLA, spremljanje, poročanje, upravljanje z incidenti, upravljanje z energijo ter upravljanje z najemi). Komponente nadzorniškega sloja ponujajo sistemom oblaka osnovni nabor kontrolnih funkcionalnosti, kot so upravljanje SLA-jev, spremljanje SLA-jev, merjenje porabe, nadzor nad politikami, storitev za obveščanje ter orkestracija. Cilj storitev z dodano vrednostjo je ponuditi komponente, ki so komplementarne sloju jedrnih storitev (tj. območja razložljivosti, podpora HA, podpora hibridnemu oblaku, migracija v času izvajanja, podpora za portabilnost, kontekstualizacija slik ter podpora virtualnim namenskim napravam. Bolj podroben opis posameznega sloja in korespondenčnih komponent je prisoten v poglavju 3.4.

Taksonomijo evalviramo, tako da ovrednotimo pet odprto-kodnih in pet komercialnih platform IaaS, ter izvedemo preslikavo njihovih funkcionalnosti v komponente in sloje, definirane v predlagani taksonomiji. Rezultati evalvacije so predstavljeni v tabeli 3.1, kjer vrednost ”X” pomeni prisotnost podpore za specifične zmožnosti posa-



Slika 3.2

Povprečna pokritost produktov IaaS glede na taksonomijo.

meznega produkta, ki na tak način ustreza naši taksonomiji, in vrednost "/" pomeni, da produkt nima podpore za določeno funkcionalnost. Slika 3.2 prikazuje povprečno pokritost posameznega produkta, glede na komponente in sloje predlagane taksonomije. Kot lahko opazimo projekta VMware vCloud in Microsoft Private Cloud spadata na sam vrh tehnološke zrelosti privatnih oblakov IaaS, medtem ko sta odprto-kodna projekta Nimbus in Eucalyptus tehnološko najmanj zrela.

3.3 Arhitekturno ogrodje IaaS

S taksonomijo smo identificirali in klasificirali fundamentalne komponente IaaS v urejene kategorije/sloje. Da bi definirali konceptualno strukturo in organizacijo slojev in komponent ter definirali odvisnosti med njimi, oblikujemo arhitekturno ogrodje IaaS. S tem ogrodjem:

1. organiziramo konceptualne sloje/komponente v arhitekturno ogrodje,
2. ponudimo skupno izhodišče za analizo, primerjavo in evalvacijo implementacijskih arhitektur IaaS,
3. definiramo odvisnosti med posameznimi sloji in komponentami.

Iz tega razloga arhitekturno ogrodje IaaS predstavlja popoln nabor funkcionalnosti holistične arhitekture IaaS (slika 3.3). V nadaljevanju (poglavje 3.4) najprej podrobno opišemo posamezne sloje in komponente, v poglavju 3.5 izvedemo diskusijo

Tabela 3.1

Preslikava predlagane taksonomije nad izbranimi sistemi IaaS.

Taksonomija	*OpenNebula*	*Eucalyptus*	*OpenStack*	*Nimbus*	*Apache CloudStack*	*AWS*	*Microsoft Private Cloud*	*VMware vCloud*	*Cisco USD*	*Oracle IaaS*
Sloj abstrakcije virov										
• Compute	X	X	X	X	X	X	X	X	X	X
• Storage	X	X	X	X	X	X	/	/	/	/
• Volume	X	X	X	X	X	X	X	X	X	X
• Omrežje	X	X	X	X	X	X	X	X	X	X
Sloj jedrnih storitev										
• Identiteta	X	/	X	/	X	X	X	X	X	X
• Razporejanje	X	X	X	X	X	X	X	X	X	X
• Repozitorij slik	X	X	X	X	X	X	X	X	X	X
• Zaračunavanje	/	/	/	/	X	X	X	X	X	X
• Logiranje	X	X	X	X	X	X	X	X	X	X
Podporni sloj										
• Sporočilno vodilo	X	X	X	X	X	X	X	X	X	X
• Podatkovna baza	X	X	X	X	X	X	X	X	X	X
• Storitev za prenos	X	X	X	X	X	X	X	X	X	X
Sloj upravljanja										
• Upravljanje z viri	X	X	X	X	X	X	X	X	X	X
• Upravljanje s federacijo	/	/	/	/	X	/	X	X	X	/
• Upravljanje elastičnosti	/	/	/	/	X	X	X	X	X	X
• Upravljanje uporabnikov/skupin	X	X	X	X	X	X	X	X	X	X
• SLA definicija	X	X	X	X	/	X	X	X	/	X
• Spremljanje	X	X	X	X	X	X	X	X	X	X
• Poročanje	/	/	/	/	X	X	X	X	X	X
• Upravljanje z incidenti	/	/	/	/	X	/	X	X	X	X
• Upravljanje z energijo	/	/	/	/	/	/	X	X	X	X
• Upravljanje z najemi	X	/	/	/	/	/	X	X	X	X
Upravljavska orodja										
• CLI orodja	X	X	X	X	X	X	X	X	X	X
• API-ji	X	X	X	X	X	X	X	X	X	X
• Nadzorna plošča	X	X	X	/	X	X	X	X	X	X
• Orkestrator	/	/	/	/	/	/	X	X	X	/
Sloj varnosti										
• Avtentikacija	X	X	X	X	X	X	X	X	X	X
• Avtorizacija	X	X	X	X	X	X	X	X	X	X
• Varnostne skupine	X	X	X	X	X	X	X	X	X	X
• Enotna prijava (SSO)	/	/	/	/	X	X	X	X	X	X
• Spremljanje varnosti	/	/	/	/	/	X	X	X	/	/
Nadzorniški sloj										
• Izvrševanje SLA-jev	/	/	/	/	/	X	X	X	/	X
• Spremljanje SLA-jev	/	/	/	/	/	X	X	X	/	X
• Merjenje	/	/	X	/	X	X	X	X	X	X
• Nadzor nad politikami	/	/	/	/	/	X	X	X	X	X
• Storitev za obveščanje	/	/	/	/	/	X	X	X	X	X
• Orkestracija	/	/	X	/	/	/	X	X	X	/
Storitve z dodano vrednostjo										
• Območja razpoložljivosti	/	/	/	/	X	X	X	X	X	X
• Gruča (HA)	/	/	/	/	X	X	X	X	X	X
• Podpora hibridnemu oblaku	X	/	/	X	/	/	X	X	/	/
• Migracija v času izvajanja	/	/	/	/	X	/	X	X	X	X
• Podpora za portabilnost	/	/	/	/	/	/	/	/	X	/
• Kontekstualizacija slik	/	/	/	X	/	/	/	/	/	/
• Podpora virtualnim namenskim napravam	/	/	/	/	/	/	X	X	/	/

o funkcionalnih odvisnosti med posameznimi sloji/komponentami, nakar v poglavju 3.6 naredimo evalvacijo obstoječih arhitektur IaaS.

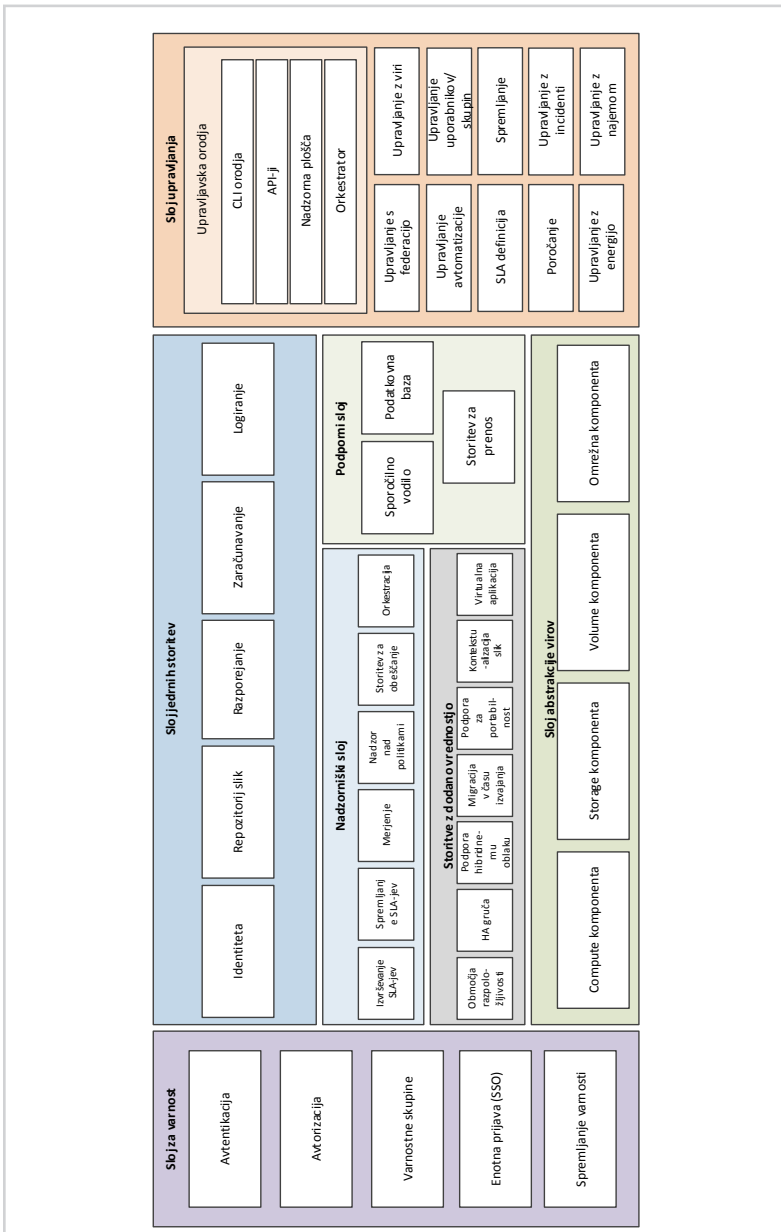
3.4 Arhitekturni sloji

3.4.1 Sloj abstrakcije virov

Sloj abstrakcije virov vključuje komponente, kot so komponente Compute, Storage, Volume ter omrežna komponenta, pri čemer vsaka igra bistveno vlogo pri infrastrukturah oblaka. Komponenta Compute temelji na programski opremi, ki je zasnovana za oskrbovanje in upravljanje virtualnih strojev, kot tudi fizičnih gostiteljev. Najbolj napredna in odmevna tehnologija Compute v industriji je Amazonov Elastic Compute Cloud (EC2). Primarna naloga te programske opreme, ki je v večini primerov realizirana kot prikriti proces (angl. Daemon), je kreiranje, terminacija ter vnovičen zagon instanc virtualnih strojev. V nekaterih primerih (na primer OpenStack Nova [105]), komponenta Compute prav tako skrbi za pripenjanje/odpenjanje komponent Volume ter pridobivanje izhodnih podatkov preko konzole. Obstajajo številni načini za upravljanje virtualnih strojev. Najbolj pogost način je z uporabo virtualizacijskega API-ja (Application Programming Interface), kot so na primer libvirt [106], XenAPI [107], vSphere API [108] in Windows Management Instrumentation (WMI) [109].

Medtem ko je Amazonov S3 [110] protokol postal "de facto" vmesnik za shrambo pri komercialnih podatkovnih oblakih [111], so se na tržišču pojavile številne odprtokodne implementacije Amazon REST API-ja. Kljub temu obstaja veliko storitev za shrambo v oblaku, ki implementirajo lastne arhitekture, ali sledijo arhitekturam drugih lastniških (angl. Proprietary) rešitev, kot je naprimer Google File System (GFS) [112]. Nekateri izmed teh odprto-kodnih storitev za shrambo podatkov v oblaku so se razvijali kot ločeni projekti [113–115], medtem ko so se drugi razvijali kot del obstoječih ogrodij IaaS [105, 116, 117]. V obeh primerih komponenta shrambe ponuja masivno, skalabilno in redundantno objektno shrambo. Še več, obstajajo orodja, kot je na primer OpenNebula [118], ki zgolj ponujajo skladišče za shrambo slik, ki omogoča nalaganje in prenašanje slik VM-jev in ne predstavlja splošno-namenske shrambe v oblaku, kot je to Amazon S3, Cumulus [111], Walrus [117, 119] ali Swift [105].

Instance VM-jev ne ponujajo persistentne shrambe (shranjeni podatki so izgubljeni, ko je instanca ponovno zagnana ali zaustavljena). Na podlagi omenjenih pomanjkljivosti, komponenta Volume omogoča upravljanje s persistentno shrambo na nivoju



Slika 3.3

Arhitekturno ogrodje IaaS.

blokov (angl. Block-level Storage), ki v večini primerov sledi implementaciji Amazonovem Elastic Block Store (EBS) [120], in se je zmožna povezovati s številnimi sistemi. Najpogostejši sistemi/protokoli, ki se jih poslužuje komponenta Volume, so Network File System (NFS), Internet Small Computer System Interface (iSCSI) in ATA over Ethernet (AoE). Nosilce (angl. Volumes) lahko kreiramo, brišemo, pripenjamo ter odpenjamo komponenti Compute, ne moremo jih deliti med več instancami, in omogočajo kreiranje posnetkov (angl. Snapshots) in hrambo le-teh v centralni sistem za shrambo (angl. Central Storage System).

Omrežje je običajno najbolj kompleksni modul vsakega sistema računalniškega oblaka. Omrežna komponenta je zadolžena za obravnavo omrežnih opravil in izvajanje sistemskih ukazov za manipulacijo omrežja. Ta opravila običajno vključujejo premostitev (angl. Bridging) vmesnikov, spreminjanje pravil pri tabelah IP naslovov (angl. Iptables Rules), samodejno dodeljevanje MAC in IP (statičnih in javnih) naslovov postavljenim VM-jem, konfiguracija omrežja za računska vozlišča, dodajanje in brisanje virtualnih omrežij, apliciranje pravil za požarne zidove na virtualne stroje itn. V številnih primerih je najbolj napredna in polno-funkcionalna omrežna nastavitev omrežje VLAN, kjer so VLAN-označeni paketki uporabljeni za ločevanje fizičnega omrežja v številna virtualna omrežja. Manjša virtualna omrežja so uporabljena za povezovanje instanc virtualnih strojev znotraj iste skupine in preprečujejo dostop virtualnim strojem, ki pripadajo drugi skupini (tj. omrežna izolacija). Ogrodja IaaS v večini primerov podpirajo številne možnosti omrežnih konfiguracij. OpenStack [105] imenuje te možnosti upravitelji omrežij (angl. Networking Managers), Eucalyptus [117, 119] pa uporablja termin omrežni načini (angl. Networking Modes).

3.4.2 Sloj jedrnih storitev

Sloj jedrnih storitev predstavlja temeljne funkcionalnosti vsakega sistema IaaS. Vključuje komponente kot so identiteta, repozitorij slik, razporejanje, zaračunavanje ter logiranje.

Osrednji del predlaganega arhitekturnega ogrodja (kot ga prikazuje slika 3.3) je sloj jedrnih storitev, ki vključuje pet komponent, tj. identiteta, razporejanje, repozitorij slik, zaračunavanje ter logiranje. Identiteta ponuja storitve za avtentikacijo in upravljanje uporabnikov, računov in vlog/pravic za različne komponente oblaka (npr. komponenta Compute, komponenta Storage ter repozitorij slik). Močno je odvisna od komponent nivoja varnosti ter od komponente nadzora nad politikami. Razporejanje

je zadolženo za pridobivanje zahteve virtualnega stroja iz vrste in določitev, na katerem računskem stroju se bo virtualni stroj izvajal. Ravno tako uporablja algoritme za zagotavljanje učinkovite uporabe fizičnih virov, v nekaterih primerih [105, 117, 118, 121] omogoča uporabnikom implementacijo lastnih razporejevalnih algoritmov, ki temeljijo na priključljivi arhitekturi (angl. Pluggable Architecture). Razporejanje mora podpirati vsaj dva osnovna pristopa:

1. *izenačevanje obremenitev* (angl. Load Balancing) – ko razporejevalni algoritem izbere naslednje vozlišče z najnižjo obremenitvijo (breme je enakomerno razporejeno med fizičnimi vozlišči),

2. *strežniška konsolidacija* (angl. Server Consolidation) – ko sistem poskuša minimalizirati število izvajajočih se fizičnih vozlišč, tako da razporeja virtualne stroje med obstoječimi alociranimi vozlišči in uporabi dodatna fizična vozlišča zgolj, ko je to nujno potrebno.

Z namenom povečanja izkoristka virov ter zmanjšanja stroškov za porabo električne energije, je s strani podatkovnih centrov uporabljena strežniška konsolidacija. Pomembna je še posebej takrat, ko so uporabniške delovne obremenitve nepredvidljive in morajo biti periodično preiskane. Vsakič, ko pride do spremembe (povečanja/pomanjšanja) zahtev uporabnika, lahko VM-jem spremenimo velikost oz. jih migriramo na druge fizične strežnike [122, 123].

Pri računalniških oblakih ima repozitorij slik pomembno vlogo, saj ponuja katalog storitev za shranjevanje in poizvedbo po virtualnih diskovnih slikah (angl. Virtual Disk Images). Vsako ogrodje ponuja lasten seznam podprtih diskovnih formatov, pri čemer so najpogosteje uporabljeni formati Raw, VHD, qcow2, VMDK, VDI itn. Takšna raznolikost podprtih formatov lahko predstavlja težave v primerih, ko prenašamo slike med različnimi namestitvami, ki so vezane na eno samo implementacijo. Kar pomeni, da če različni ponudniki ponujajo različna jedra, kar je običajno praksa, lahko slika, ki deluje na enem jedru, zaradi integracijskih težav preneha delovati na drugem jedru [100].

Vsak VM v oblaku se izvaja glede na razpoložljivost in performanco SLA-jev, ki so definirani med ponudnikom oblaka in njegovim uporabnikom. Metrike, ki so lahko uporabljene za zaračunavanje VM-jev so definirane za posamezno storitev oblaka. Te metrike so lahko običajne upravljavske akcije, kot so zagon, zaustavitev in vnovični zagon VM-ja. Komponenta zaračunavanja lahko posreduje dogodke logov, odmerja loge

ter ustvarja ustrezne račune. Zraven tega upravlja s podatki stranke, pošilja poročila zaračunavanja, procesira prejeta plačila, sledi oz. spremlja izdane račune itn. Politike zaračunavanja (angl. Billing Policies) nadzirajo ocenjene stroške za porabljene vire, saj so zaračunani glede na alokacijo virtualiziranih virov, na primer virtualni CPU, pomnilnik in shramba.

Vsak sistem računalniškega oblaka mora imeti nastavljen primeren nivo logiranja (angl. Logging), da bi ustrezal zahtevam organizacije po sledenju (angl. Audit) [124]. Komponenta logiranja mora iz tega razloga slediti celotnemu naboru operacij oblaka, ki naslavljajo varnost in zmogljivost. Pri tem operacije oblaka onemogočajo uporabo virov oblaka, ne da bi s tem puščali sled, tudi takrat ko se sistem upravlja s pravicami administratorja. Posledično mora vsak oblak IaaS ponujati sledljive loge (angl. Auditing Logs) za različne operacije, kot so skaliranje VM-jev ter administracija IP-jev. Za zagotavljanje celovitosti sledenja logov so nadvse pomembne stroge zahteve po logiranju. Ponudnik, ki uporablja platformo IaaS, mora ponuditi dokazila, kako bo zadostil svojim omejitvam, medtem ko mora poskrbeti za skladnost z regulativnimi in zakonodajnimi restrikcijami.

3.4.3 Podporni sloj

Ta sloj služi kot vmesni sloj, preko katerega so ostali sloji v interakciji in med seboj komunicirajo. Podporni sloj obravnavamo kot podporni nivo sloja jedrnih storitev in predstavlja ključnega posrednika med slojem jedrnih storitev ter slojem abstrakcije virov. Vključuje sporočilno vodilo, podatkovno bazo ter storitev za prenos.

Sporočilno vodilo ponuja osrednje vozlišče za posredovanje sporočil med različnimi storitvami oblaka in običajno uporablja tehnologije, ki temeljijo na protokolu Advanced Message Queuing Protocol (AMQP) [125] ali Java Message Service (JMS) (npr. RabbitMQ in ActiveMQ [126]). Dogodki, prisotni v oblakih zahtevajo vodilo (angl. Bus), ki distribuira dogodke iz izvorne komponente (proizvajalca dogodkov) v ponorne komponente (potrošnike dogodkov), in mora slediti pristopu EDA (Event-Driven Architecture), saj se le-ta nanaša na produkcijo, detekcijo, uporabo in reakcijo na različne dogodke.

Podatkovna baza na drugi strani hrani večino konfiguracijskega in izvajalnega stanja infrastrukture oblaka, vključno z razpoložljivimi instancami VM-jev, trenutno uporabljenih instanc VM-jev, razpoložljivimi omrežji, nabora "par-ključev" (angl. KeyPairs), metapodatki slik itn. OpenStack na primer uporablja različne tipe sporočilnih

vrst, da bi pospešil komunikacijo med različnimi prikritimi procesi (tj. topics queues, fanout queues ter host queues). Aplikabilnost relacijskih podatkovnih baz je celovito prisotna v ogrodjih IaaS [105, 116–118, 127–129], pri čemer vsako ogrodje uporablja drugačen nabor tehnologij. OpenNebula na primer podpira zgolj podatkovni bazi sqlite3 ter MySQL, OpenStack uporablja knjižnico SQLAlchemy [130] kot abstrakcijo za dostop do podatkovne baze, saj omogoča uporabo poljubne relacijske baze, ki jo ta knjižnica podpira, medtem ko VMware vCloud [128] podpira podatkovni bazi Oracle Database ter MSSQL Server. Storitve oblaka si tipično izmenjujejo informacije preko sporočilnih vrst in podatkovne baze, da bi prestregle zahteve API-jev ter izvršile zahtevana opravila. Ker so virtualni stroji sestavljeni iz slik, ki hranijo njihove virtualizirane trde diske, morajo biti ti diski preneseni v gručo vozlišč (angl. Cluster Nodes), da bi se virtualni stroji lahko izvajali brez degradirane performance. Storitev prenosa prevzame to odgovornost, medtem ko je zadolžena za vse prenose datotek, ki so potrebne za pravilno postavitev virtualnih strojev, vključno s prenosom slik do vozlišča gruče, ki je izbrano za izvajanje virtualnega stroja, in prenosom iz vozlišča gruče v repozitorij slik.

3.4.4 Sloj varnosti

Varnost je eden izmen najpomembnejših aspektov, ki ga naslovijo organizacije IT, preden razmišljajo o prisvojitvi infrastrukturnih oblakov. Pravzaprav je večina groženj, s katerimi se soočajo oblaki IaaS, že znanih iz drugih področij računalništva, vendar kljub temu oblaki dodatno vpeljujejo nekatere karakteristike (npr. skalabilnost, zasebnost podatkov in večnajemništvo), ki prinašajo nove varnostne skrbi [131].

V disertaciji opravimo celovit pregled literature, vezane na varnostni aspekt oblakov IaaS [14, 16, 131–133] in posledično klasificiramo sloj varnosti v pet komponent: avtentikacija, avtorizacija, varnostne skupine, enotna prijava (SSO) ter spremljanje varnosti, ki skupaj predstavljajo pet fundamentalnih varnostnih aspektov arhitekture IaaS. Sloj varnosti v našem primeru zagotavlja varnost na nivoju infrastrukture. Avtentikacija je običajno realizirana preko nadzora dostopa, ki temelji na žetonih (angl. Token-Based Access Control) in uporablja mehanizme, kot so LDAP (Lightweight Directory Access Protocol), uporabniška imena in gesla, pari ključev SSH (angl. SSH Keypairs), Kerberos ali X509 certifikati. To komponento uporabljamo za avtentikacijo uporabnikov za dostop do infrastrukture oblakov, kot tudi za dostop do oskrbljenih VM-jev.

Komponenta za avtorizacijo avtorizira avtenticiranega uporabnika za specifična opravila (npr. običajni uporabnik oblaka ne more kreirati novih virtualnih omrežij, pri če-

mer administrator oblaka to lahko). Varnostne skupine definirajo pravila za požarne zidove, ki jih apliciramo nad instancami VM-jev, ki so asociirani s posamezno skupino. Kar pomeni, da instance, ki pripadajo isti skupini, pripadajo tudi istemu podomrežju in niso vidne instancam iz ostalih skupin. Enotna prijava (SSO) je asociirana s tehnologijo in naborom industrijskih standardov, ki se imenuje federacija identitet, saj omogoča varno deljenje identitet med ločenimi omrežji in heterogenimi računalniškimi viri [134]. Ker različne platforme IaaS podpirajo različne avtentikacijske mehanizme, mora SSO interno prevajati in hraniti različne poverilnice (angl. Credentials) v primerjavi s tem, kar je uporabljeno pri prvotni avtentikaciji. Vendar pa za razliko od elektroenergetskega sistema, telekomunikacijskega sistema in drugih storitveno usmerjenih služb, oblaki še zmeraj ne morejo federirati in interoperati v takšnem obsegu, kar predstavlja priložnost za nadaljnji razvoj in raziskave na področju računalniških oblakov [28, 135, 136]. Nazadnje, komponenta spremljanje varnosti implementira spremljanje omrežne varnosti, revizijske sledi in druga orodja, ki ponujajo senzorje za detekcijo varnostnih dogodkov [132]. Mehanizmi spremljanja v času izvajanja preverjajo nabor pogojev opredeljenih s politikami (angl. Policy-Defined Conditions), ki so definirane znotraj komponente nadzora nad politikami iz nadzorniškega sloja, da bi odkrili grožnje ali anomalije v obnašanju storitev oblaka. V takšnem življenjskem ciklu varnostnih politik, mora varnostni arhitekt ustvariti, izvrševati in upravljati odločitvene politike, ki so običajno dinamične. Komponenta spremljanja varnosti na tak način združuje zmožnost aktivnega spremljanja in varovanja gostujočih VM-jev, ne da bi pri tem morala nameščati varnostno kodo znotraj samega VM-ja [133]. Spremljanje varnosti je doseženo z uporabo varnih metod za izvajanje programske opreme, ki so skladne z definiranimi politikami. Takšno samodejno izvrševanje politik v času izvajanja je kritičnega pomena za doseganje primerne stopnje varnosti v virtualiziranih platformah IaaS.

3.4.5 Sloj upravljanja

Sloj upravljanja je zadolžen za celovito upravljanje z infrastrukturnim, računalniškim oblakom. Ta sloj vključuje enajst komponent: upravljavska orodja, upravljanje s federacijo, upravljanje elastičnosti, upravljanje z viri, upravljanje uporabnikov/skupin, SLA definicijo, poročanje, spremljanje, upravljanje z incidenti, upravljanje z energijo ter upravljanje z najemi.

Za zagotovitev interakcije s spodaj ležečimi storitvami in viri, upravljavska orod-

ja specificirajo API-je, orodja, grafične uporabniške vmesnike in orkestratorje oblaka. Vmesnik z ukazno vrstico (angl. Command-Line Interface – CLI) služi kot ključna točka interakcije s sistemom in njegovo spodaj ležečo arhitekturo. Z uporabo orodij CLI lahko administratorji oblaka vršijo številne ukaze za nadziranje in spremljanje virtualnih strojev, dodajanje, brisanje in spremljanje gostiteljev ali virtualnih omrežij itn. Zelo pomembno je, da orodja CLI vključujejo popoln nabor ukazov, saj se zna zgoditi, da je dostop do oblaka na osnovi lupine (angl. Shell-Based Access) edini uporabniški vmesnik do oblaka. Na primer, ko je oblak prvotno nameščen, ali ko uporaba alternativnih uporabniških vmesnikov (npr. nadzorno ploščo ali programski API) ni možna, so orodja CLI edini način za komunikacijo z oblakom. API se uporablja za programski dostop do infrastrukture oblaka, npr. preko protokola Representational State Transfer (REST). Prav tako omogoča avtomatizacijo in razširljivost opravil, ki so specifična za oblak. Vmesnik REST API ter vmesnik CLI lahko uporabljajo isto jedro API-ja, ki ni izpostavljeno administratorju oblaka in je uporabljeno zgolj interno s strani upravljavskih orodij. Iz tega razloga uporabljajo sistemi IaaS različne tipe API-jev, ki temeljijo na različnih standardih, kot so REST, Simple Object Access Protocol (SOAP), OGF Open Cloud Computing Interface (OCCI), XML-RPC, Web Services Resource Framework (WSRF) itn. Zraven lokalnih API-jev (angl. Native APIs), večina ogrodij ponuja podporo za "de facto" Amazon API-je, kot sta na primer EC2 API in S3 API. Večina rešitev IaaS ponuja nadzorno ploščo, ki običajno vsebuje podmnožico produktnih funkcionalnosti in je namenjena tako administratorjem oblaka, kot tudi končnim uporabnikom, pri čemer poenostavlja tipične upravljavske operacije v okolju oblaka. Zaradi nezrelosti trenutnih nadzornih plošč (npr. Horizon in Sunstone), upravljanje virtualnih in fizičnih virov še vedno ne dosega enake stopnje, kot jo lahko dosežemo z uporabo orodij z ukazno vrstico. Orkestrator na drugi strani ponuja orodje za upravljanje delovnih tokov (angl. Workflow Management Tool), ki avtomatizira ustvarjanje, spremljanje in postavitev heterogenih virov in procesov.

Upravljanje s federacijo in elastičnostjo sta dve izmed esencialnih upravljavskih komponent holistične arhitekture IaaS. Upravljanje s federacijo je kompleksna in zahtevna komponenta, ki je zadolžena za medsebojno povezovanje (integracijo) različnih virov oblaka in temelji na mehanizmu SSO. Federirani oblaki (angl. Federated Clouds) zahtevajo enotne upravljavske vmesnike za VM-je, na katerih se izvajajo storitve oblaka. Omenjena komponenta mora omogočati upravljanje z več oblaki (angl. Multi-Cloud Management), ki izbira med lokacijami oblakov in ponuja enotne storitve oblaka, kot

so storitveni katalog federacije (angl. Service Catalog Federation), kolaborativno upravljanje (angl. Collaborative Management) in storitve za migracijo virtualnih strežnikov aplikacij (angl. Application Virtual Server Migration Services). Upravljanje z elastičnostjo po drugi strani uporablja napredne tehnologije za avtomatizirano in dinamično obravnavo virov, ki temeljijo na uporabniško definiranih politikah (angl. User-Defined Policies) [137]. Najbolj prominenten primer takšne tehnologije je Amazonov Auto Scaling, ki omogoča horizontalno skaliranje kapacitet EC2 glede na uporabniško definirane pogoje. Upravljanje SLA-jev vključuje: definicijo SLA (osnovna shema s parametri QoS), spremljanjem SLA-jev ter izvrševanje SLA-jev glede na definirane politike. Definicija pogodbe SLA ustreza upravljavskemu sloju, medtem ko sta spremljanje in izvrševanje postavljena v nadzorniški sloj. SLA predstavlja dokument, ki vključuje opise dogovorjenih storitev, parametrov storitvenega nivoja (angl. Service Level Parameters), zagotovila, akcije ter pravna sredstva za vse primere kršitev dogovora. Iz tega razloga je dobra definicija SLA izrednega pomena, saj služi kot pogodba med ponudnikom in končnim uporabnikom, pri čemer arhitekture IaaS zahtevajo prisotnost podpori za takšno obliko pogajanja. Čeprav vsi večji ponudniki, kot so Amazon EC2, ElasticHosts, GoGrid, in TerraMark, ponujajo zmožnosti definicije SLA, nobeno izmed zadnjih verzij odprto-kodnih ogrodij ne podpira vgrajene podore za pogajanje SLA-jev. Kot posledica tega so nastala številna eksterna ogrodja, ki jih lahko integriramo s sistemi IaaS. Eden izmed primerov je ogrodje SLA@SOI, ki je bilo razvito kot del projekta European FP7 ICT Integrated Project, in je bilo aplicirano že na številnih rešitvah (npr. modificirana verzija ogrodja Tashi [138]).

Komponenta za poročanje je zadolžena za generiranje stroškovnih poročil (angl. Cost Reports), poročil porabe (angl. Usage Reports) ter primerjalnih poročil za različne uporabnike, ki jih lahko pri generiranju poročil primerjamo z določenim stroškovnim modelom (angl. Cost Models). Še več, ta komponenta mora ponujati podporo za standardna poročila, ki vsebujejo kapaciteto, izkoriščenost ter druge sistemske metrike in služijo kot temelj za zaračunavanje uporabnikom. Večina rešitev ravno tako omogoča izvoz poročil v standardiziran format XML. Skladno s tem lahko v nadaljevanju razvijalci uporabijo transformacije XSLT za transformiranje grobega formata XML (angl. Raw XML) v format, ki ga podpira strankin sistem zaračunavanja (angl. Billing System). Poročila se lahko uporabljajo preko ponudnikovega uporabniškega vmesnika ali z uporabo API-jev poročanja (angl. Reporting APIs) ter so na voljo v različnih formatih (na primer PDF, XLS, XML itn.). Upravljanje z energijo daje administratorjem oblaka

nadzor nad porabo energije, pri čemer ponuja funkcionalnosti, kot so na primer:

- Definiranje politik za upravljanje z energijo (angl. Power Management Policies), za uveljavitev različnih nastavitev za periode "peak" ter "non-peak" uporabniških aktivnosti.
- Spremljanje VM-jev/gostiteljev in uporabniških aktivnosti za zmanjševanje nenamerne prekinitve uporabnikov.
- Poročanje o porabi energije in stroškovnih prihrankih.

Upravljanje z viri v večini primerov naslavlja upravljanje z VM-ji ter gostitelji, vendar tudi vključuje upravljanje drugih virov oblaka. Upravljanje z VM-ji je zadolženo za upravljanje, kreacijo, prekinitev ter terminiranje virtualnih instanc, pri čemer je upravljanje z gostitelji zadolženo za upravljanje s fizičnimi strežniki. Ta komponenta je odvisna od sloja abstrakcije virov, saj je zadolžena za upravljanje vseh njenih komponent. Najbolj pogoste funkcionalnosti komponente upravljanja z uporabniki in skupinami so dodajanje in odstranjevanje uporabnikov, dodajanje in odstranjevanje skupin ter dodajanje uporabnikov v določeno skupino. Glede komponente za spremljanje, imajo različni postavitveni modeli oblaka (privatni ali javni oblak) različne potrebe in zahtevajo drugačne pristope. Javni oblaki so običajno geografsko razpršeni bazeni virov, ki zahtevajo večje investicije v spremljanje prometa in zagotavljanja skalabilnosti. Zraven tega, večina odprto-kodnih arhitektur ne podpira vgrajene zmožnosti za spremljanje. Kljub temu obstajajo odprto-kodni projekti, kot sta Ganglia [139] in Nagios [140], ki ju lahko integriramo s sistemi IaaS. Zato je komponenta za spremljanje tesno vezana na sloj abstrakcije virov, saj od njih pridobiva relevantne metrike (poraba CPU, pomnilnik, število izvajajočih se VM-jev itn.) s pomočjo senzorjev na fizičnih vozliščih. Poleg tega se v primeru prekinitve oz. prenehanja delovanja storitve, upravljanje z incidenti osredotoča na obnovitev izvajanja storitev oblaka v dogovorjenem roku, kot tudi infrastrukture, na kateri se izvaja. Incident lahko definiramo kot parcialna oziroma popolna nerazpoložljivost virov in storitev. Zato je obnova sistema in infrastrukture po nastalem incidentu (prekinitvi vitalnih operacij), pomemben mehanizem sistemov IaaS. Navsezadnje je komponenta upravljanja najema pristojna za dodeljevanje najemnih pogodb (angl. Leases) za vire uporabnikov, ki jih je mogoče vnovčiti na neki točki v prihodnosti. Ta zmožnost omogoča uporabnikom, da zahtevajo ekskluzivno uporabo

teh virov, ki so lahko opisani na različne načine, na primer "Potrebujem 5 vozlišč, vsako s 3 CPU-ji in 32GB pomnilnika, od 13h do 19h jutri", kot to počne odprto-kodni projekt Haizea [141] ter številne druge komercialne rešitve [128, 129, 142, 143].

3.4.6 Nadzorniški sloj

Nadzorniški sloj zajema osnovne funkcije nadzora v arhitekturah IaaS, vključno z izvrševanjem SLA-jev, spremljanjem SLA-jev, merjenjem porabe, nadzorom nad politikami, storitvijo za obveščanje, ter komponento za orkestracijo. V splošnem, komponente nadzorniškega sloja omogočajo nadzor nad infrastrukturnimi viri in storitvami. V sistemih IaaS so ponujeni viri, ki so skladni z naborom vnaprej definiranih nefunkcionalnih lastnosti, specificirani in izpogajani v obliki SLA-jev. Ko je dokument SLA enkrat izpogajan, je potrebno zagotoviti realno-časovno spremljanje in izvrševanje dogovorjenih nivojev storitve. Komponenta spremljanja SLA-jev definira načine, na katere so lahko parametri storitve spremljani, ter format poročil spremljanja (angl. Monitoring Reports). Izvrševanje SLA-jev na drugi strani poskrbi za izvedbo dogovorjeni parametrov QoS z minimalnimi kršitvami predpisanimi v dokumentu SLA. Da bi se izognili dragim kršitvam SLA in da bi lahko pravočasno reagirali na napake in na spremembe v okolju, je v arhitekturo IaaS potrebno vpeljati strategije za uzakonitev SLA-jev (angl. SLA Enactment Strategies). Te strategije vključujejo primerne koncepte za spremljanje virov oblaka.

Komponenta merjenja virov je bistvenega pomena za natančno merjenje uporabnikove porabe ter oblikovanje uporabnikovega obnašanja s t. i. politikami zaračunavanja, ki omogočajo stroškovno transparentnost in odgovornost v okolju oblaka. Komponenta merjenja omogoča odmerjanje, analiziranje ter poročanje uporabe virov v privatnih in javnih okoljih (npr. shramba, procesiranje, pasovna širina, aktivni uporabniki in računi). Prav tako omogoča ponudnikom oblaka validacijo in prilagajanje finančnih modelov glede na zahteve po določenih virih. Primer takšnih sistemov sta System Center Operations Manager [144] ter vCenter Operations Manager [145]. Nadzor nad politikami vključuje funkcionalnosti, kot so nadziranje kvot, nadziranje zasebnosti podatkov, nadziranje konfiguracij programske opreme, strojne opreme itn. Te politike obvladujejo obnašanje končnih uporabnikov računalniških oblakov.

Storitev za obveščanje omogoča operiranje, kostumiziranje ter pošiljanje notifikacij različnim deležnikom oblaka (npr. končnim uporabnikom in administratorjem). Notifikacije so lahko poslane iz različnih razlogov:

- ko je poslana zahteva oz. potrebna potrditev,
- notifikacija ob napakah in opozorilih,
- ob registraciji novega uporabnika,
- ob kršitvi dogovorjenega nivoja SLA,
- ob pomanjkanju fizičnih virov,
- opozorilo, preden se pojavi kršitev SLA itn.

Komponenta za orkestracijo predstavlja orkestracijsko platformo, ki ponuja katalog razširljivih delovnih tokov za izdelavo in izvajanje avtomatiziranih, konfigurabilnih procesov za upravljanje infrastrukture oblaka, kot tudi drugih zunanjih tehnologij. Takšna platforma v večini primerov ponuja [56, 59–61]:

- Vizualno okolje "drag-and-drop" za delovne tokove, ki omogoča izgradnjo zaporedja operacij v logične tokove.
- Notifikacije opozoril in incidentov, ki se poslužuje storitve za obveščanje.
- Pogon za politike (angl. Policy Engine), ki omogoča spremljanje in generiranje dogodkov, da bi dosegli spreminjajoče se pogoje v heterogenih okoljih.
- Integracijske pakete in adapterje za povezavo s sistemi/storitvami oblaka različnih ponudnikov.
- Skriptni pogon za izdelavo novih gradbenih blokov (integracijskih paketov ali adapterjev) za komponento orkestracije.

3.4.7 Storitve z dodano vrednostjo

Storitve z dodano vrednostjo so napredne funkcionalnosti, ki predstavljajo ključni diferenciator pred ostalimi tehnologijami. Primarni cilj tega nivoja je ponuditi komponente, ki so komplementarne komponentam iz sloja jedrnih storitev. Storitve, ki predstavljajo dodano vrednost sistemov IaaS so območja razpoložljivosti, gruča HA, podpora hibridnemu oblaku, migracija v času izvajanja, podpora za portabilnost, kontekstualizacija slik ter virtualna namenska naprava.

Prva komponenta t. i. območje razpoložljivosti, omogoča visoko stopnjo razpoložljivosti med različnimi geografskimi regijami, brez potreb po dodatnih modifikacijah pri postavitvi ter dodatnih strategij povezanih z replikacijo podatkov. Uporaba redundantnega območja razpoložljivosti pomeni imeti redundanten podatkovni center, saj je vsako območje popolnoma ločeno od ostalih območij znotraj iste regije. Najbolj znan industrijski predstavnik te komponente je Amazon Multi-AZ [146]. Komponenta je primarno zasnovana za zagotavljanje visoke stopnje razpoložljivosti storitev v oblaku, ki je zagotovljena z gručo HA. Ta komponenta omogoča konfiguracijo redundantnih strežnikov, ki so uporabljeni za ponujanje storitev v primeru prekinitev sistema, in mora biti sposobna zaznave napak strojne in programske opreme ter ponovno zagnati aplikacijo na drugem sistemu (brez potrebe po intervenciji administratorja). Primer v industriji sta tehnologiji vSphere HA [108] ter Hyper-V HA [147].

Podpora hibridnemu oblaku omogoča razširitev računalniških virov na zunanji, (običajno) javni sistem IaaS. Mehanizem migracije virtualnega stroja oz. storitve poteka iz preobremenjenega gostitelja na fizični strežnik v drugem oblaku, mora zagotavljati določeno stopnjo performance v heterogenem okolju IaaS. Primer tehnologij, ki zagotavljajo podporo hibridnemu oblaku, so VMware vCloud Connector [148], System Center 2012 App Controller [149] ter različna ogrodja "multi-cloud", kot je na primer jclouds [87].

Migracija v času izvajanja zahteva v sistemu IaaS kot predpogoj konfigurirano gručo HA in na tak način zagotavlja mehanizem preklopa (angl. Failover). Omogoča transparentno premikanje izvajajočih se virtualnih strojev iz enega fizičnega vozlišča gruče do drugega, brez padle omrežne povezave in prekinitve delovanja (angl. Downtime). Običajno sta podprta dva tipa migracije v času izvajanja:

1. migracija Compute v času izvajanja (stanje VM-ja se shrani v pomnilnik),
2. migracija Storage v času izvajanja (prekopira se celoten virtualni disk).

Težave z interoperabilnostjo v okviru IaaS se pojavijo, ko različni sistemi oblaka poskušajo sodelovati, tako da si izmenjujejo virtualne stroje (tj. federacija). V primeru migracije podatkov in izvajanja aplikacij na novem oblaku IaaS, je potrebno zajeti sliko virtualnega stroja, ter jo migrirati k drugemu ponudniku IaaS, ki morebiti uporablja drugačno virtualizacijsko tehnologijo [12]. Komponenta za portabilnost mora zagotavljati tri stvari:

1. ponujati mehanizem za pretvarjanje med različnimi diskovnimi formati, da bi omogočili transparentno premikanje diskovnih slik med različnimi namestitvami IaaS,

2. zagotavljati hipervizor-agnostično podporo med različnimi sistemi IaaS. Posamezen IaaS sitem mora ponujati podporo za različne tipe virtualizacijskih tehnologij (Xen, KVM, VMware, Hyper-V itn.),

3. podpirati standardizirane API-je tipa "multi-cloud", ki naslavljajo programabilnost infrastrukturnih virov.

Zaradi pomanjkanja ustreznih tehnologij in odprtih standardov je sprejetje te vizije v današnjih časih še vedno ovirano, saj obstoječe rešitve računalništva v oblaku niso bile zasnovane v skladu z interoperabilnostjo [150]. Skladno s potrebami interoperabilnih vmesnikov API v primeru računalništva v oblaku, obstajajo precejšnja prizadevanja za določitev standardov na ravni IaaS [112], kot sta na primer OCCI ali Cloud Data Management Interface (CDMI). Open Virtualization Format (OVF), še ena pobuda za standardizacijo na nivoju IaaS, predstavlja dobro rešitev za prenašanje virtualnih strojev med oblaki (tj. uvoz in izvoz virtualnih strojev). Poleg tega so nekatera prizadevanja za vzpostavitev interoperabilne in zvezne oblake že bila dosežena v evropskih projektih tipa FP7, kot so RESERVOIR [28], Contrail [151] in mOSAIC [135].

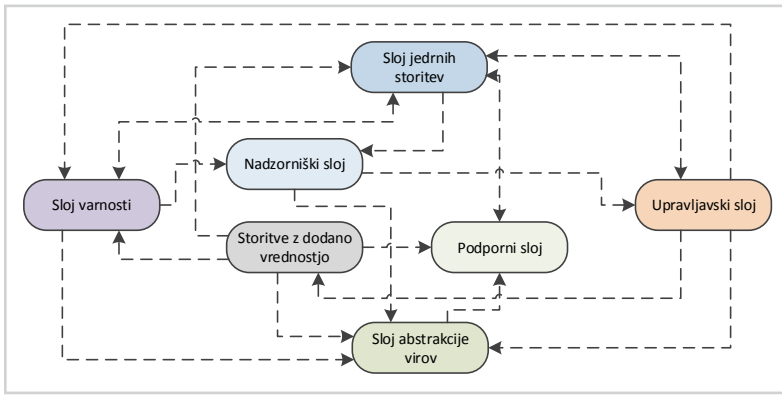
Kontekstualizacija slik omogoča postavitev instanc VM glede na deljeno sliko, ki je kostumizirana z informacijami (običajno ob zagonu) in na tak način omogoča postavitev instance v specifični kontekst. Na primer, razvijalec želi oskrbeti VM s kontekstualizirano podatkovno bazo preko portala oblaka. Pri tem mora na portalu specificirati parametre, ki so specifični za to bazo. Tipični parametri za konfiguracijo podatkovne baze so številka vrat (angl. Port Number), geslo administratorja, ime podatkovne baze, uporabniško ime podatkovne baze in geslo podatkovne baze. Ti parametri se uporabijo za kostumizacijo VM-ja, takoj za tem, ko je bil le-ta ustvarjen iz deljene predloge (angl. Shared Template). Kostumiziran VM ponuja razvijalcem personalizirano instanco podatkovne baze, ki je postavljena v specifični kontekst. Zaradi nekompatibilnosti mehanizmov za kontekstualizacijo, lahko virtualne slike delujejo pri enem ponudniku in odpovejo pri drugem [100]. Najbolj odmevna rešitev, ki naslavlja kontekstualizacijo je Nimbus Context Broker [116], spletna storitev, ki pospešuje varno izmenjavo kontekstno-specifičnih informacij (na primer, naslovi IP ali ključi SSH)

med več postavljenimi storitvami oblaka [152]. Iz tega razloga mora kontekstualizacija slik ponujati abstrakcijo (virtualno namensko napravo), iz katere lahko izpeljemo vse možne implementacije [153]. Na primer, sledimo strukturi metapodatkov, kot je to storjeno pri oblaku EC2, ali s pripenjanjem (angl. Mounting) datotečnega sistema, kot je to realizirano pri OVF. Čeprav večina trenutnih rešitev še nima te sposobnosti, se kontekstualizacijo slik obravnava kot ključno komponento sistemov IaaS. Zadnja komponenta storitev z dodano vrednostjo, virtualna aplikacija, je večkrat poimenovana tudi kot storitev oblaka (angl. Cloud Service) ali načrt storitve (angl. Service Blueprint). Temelji na specifičnih vsebnikih (angl. Containers), ki omogočajo načrtovanje in konfiguriranje večslojnih aplikacij.

3.5 Funkcijske odvisnosti

V tem poglavju so opisane funkcijske odvisnosti med posameznimi sloji. Nadzorniški sloj, storitve z dodano vrednostjo, sloj upravljanja ter sloj varnosti kažejo odvisnosti od sestavnih delov sloja abstrakcije virov. Ker komponente iz sloja abstrakcije virov (komponente Compute, Storage, Volume ter omrežje) predstavljajo temelj vsakega sistema IaaS, se večina komponent iz drugih slojev zanaša na vsaj eno izmed komponent sloja abstrakcije virov. Na primer, varnostne skupine iz sloja varnosti so odvisne od omrežne komponente sloja abstrakcije virov, saj varno ločuje (segregira) instance VM med različnimi podomrežji in jih na tak način naredi nevidne instancam iz drugih skupin. Sloj abstrakcije virov se zanaša na podporni sloj, preko katerega posredno komunicira s slojem jedrnih storitev, kot prikazuje slika 3.4. Na primer, storitve oblaka si izmenjujejo informacije preko sporočilnega vodila in podatkovne baze z namenom izvršitve zahtev API-jev, ki so sprožene s strani komponent iz sloja jedrnih storitev, in izvedejo zatavana opravila vezana na komponente iz sloja abstrakcije virov.

Sloj varnosti nakazuje odvisnosti od komponent znotraj sloja jedrnih storitev, nadzorniškega sloja in sloja virov abstrakcije, pri čemer sloj jedrnih storitev, storitve z dodano vrednostjo in sloj upravljanja kažejo odvisnosti od sloja varnosti. Na primer, komponenta identiteta (tj. storitev identitet) iz sloja jedrnih storitev ponuja storitve za avtentikacijo uporabnikov, uporabniških računov in pravic/vlog za različne komponente oblaka ter njihove storitve, saj je močno odvisna od komponent sloja varnosti, kot so avtentikacija, avtorizacija ter SSO. Storitve z dodano vrednostjo so odvisne od sloja jedrnih storitev, sloja varnosti, podpornega sloja ter sloja abstrakcije virov, pri čemer sloj upravljanja odraža odvisnosti od storitev z dodano vrednostjo. Na primer,

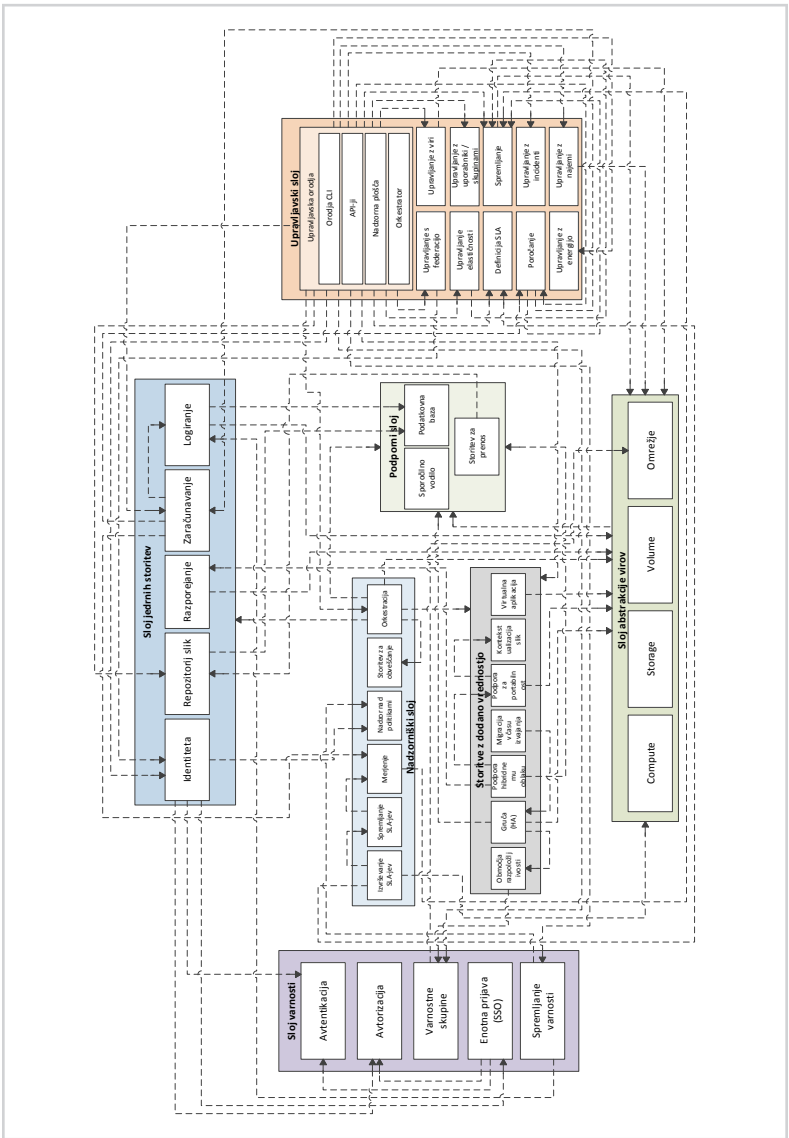


Slika 3.4

Funkcijske odvisnosti med arhitekturnimi sloji.

podpora hibridnemu oblaku se zanaša na komponento razporejanja znotraj sloja jedrnih storitev, na komponento za portabilnost znotraj istega sloja, ter na komponente znotraj podpornega sloja, z namenom zagotoviti transparentno razširitev storitev oblaka iz zasebnih okolij v javne namestitve oblakov.

Podporni sloj izkazuje odvisnosti od komponent iz sloja jedrnih storitev, medtem ko so sloj jedrnih storitev, storitve z dodano vrednostjo ter sloj abstrakcije virov odvisni od podpornega sloja. Na primer, komponente znotraj sloja jedrnih storitev, ki želijo sodelovati (biti v interakciji) s spodaj ležečimi viri, so močno odvisne od podpornega sloja, da bi uspešno izvedle svoja opravila. Iz tega razloga, sporočilno vodilo prenaša sporočila med različnimi storitvami oblaka, podatkovna baza hrani konfiguracijske podatke ter realno-časovna stanja infrastrukture oblaka, medtem kot storitev prenosa prenaša slike do gruče vozlišč. Sloj upravljanja je odvisen od komponent sloja jedrnih storitev, sloja varnosti, storitev z dodano vrednostjo ter sloja abstrakcije virov. Nasprotno sta sloj jedrnih storitev in nadzorniški sloj odvisna od komponent znotraj upravljavskega sloja. Komponenta orkestrator iz upravljavskega sloja je na primer odvisna od komponente za orkestracijo iz nadzorniškega sloja, saj ponuja orodje za upravljanje z delovnimi tokovi, ki deluje nad komponento orkestrator. Drugi primer je komponenta izvrševanja SLA-jev iz nadzorniškega sloja, ki poskrbi, da so dogovorjeni parametri QoS (definirani znotraj definicije SLA v upravljavskem sloju) izvedeni z minimalnimi kršitvami. Slika 3.5 prikazuje odvisnosti med komponentami predlaganega arhitekturnega ogrodja IaaS.



Slika 3.5

Funkcijske odvisnosti med posameznimi komponentami arhitekture IaaS.

3.6 Evalvacija obstoječih arhitektur IaaS

Predlagana taksonomija ter arhitekturno ogrodje IaaS sta uporabljena za evalvacijo različnih arhitektur IaaS v številnih realnih projektih, ki so vključevali najbolj pogoste komercialne in odprto-kodne produkte IaaS [101, 102]. V našem primeru je evalvacija izvedena preko primerjave in analize štirih odprto-kodnih rešitev: OpenNebula, Eucalyptus, OpenStack in Nimbus; ter treh komercialnih rešitev: Amazon AWS, Microsoft Private Cloud and VMware vCloud (tabela 3.2). Da bi uspešno primerjali rešitve IaaS skladno s taksonomijo in glede na komponente znotraj slojev predlaganega arhitekturnega ogrodja, smo najprej vsako platformo IaaS analizirali in testirali, pri čemer smo se zanašali na celovito literarno študijo in tehnično namestitev oz. testiranje. Primerjava pokaže funkcionalno podporo in funkcionalnosti posamezne rešitve in hkrati pokaže tudi, kako določena funkcionalnost produkta IaaS ustreza slojem in komponentam taksonomije in arhitekturnega ogrodja IaaS. Vrednost "/" v tabeli predstavlja pomanjkanje podpore za posamezne funkcionalnosti, medtem ko vrednost "Internal" predstavlja kazalnik za vgrajeno podporo funkcionalnosti, ki nima točno določenega imena. Ostale vrednosti predstavljajo konkretne tehnologije, ki zadoščajo komponentam našega predlaganega arhitekturnega ogrodja. Celovito ogrodje za evalvacijo široke palete produktov prestavlja poglavitno primerjalno osnovo, ki omogoča organizacijam IT sprejemati dobre odločitve pri adaptiranju najprimernejših tehnologij.

Evalvacija je pokazala (1) opazno razliko funkcijske podpore in zmožnosti med komercialnimi in odprto-kodnimi platformami IaaS, (2) bistveno pomanjkljivost pomembnih arhitekturnih komponent v smislu izpolnjevanja obljube infrastrukturnih oblakov ter (3) uporabnost predlagane taksonomije ter enotnega arhitekturnega ogrodja v realnem svetu, ki na tak način omogoča sprejemanje odločitev v organizacijah IT pri izbiri najbolj ustrezne rešitve IaaS.

Poleg vgrajene podpore za spremljanje in samodejnega skaliranja, odprto-kodnim orodjem primanjkuje tudi drugih funkcionalnosti, kot so podpora za virtualne (namenske) aplikacije, orkestracija, upravljanje z incidenti in energijo, zaračunavanje, logiranje, merjenje, upravljanje s SLA-ji itn. Na primer, orodja za spremljanje so v večini primerov na voljo kot zunanje rešitve (angl. Third-Party Solutions) in morajo biti integrirane z obstoječimi odpro-kodnimi rešitvami. Takšna integracija lahko predstavlja številne težave v primerih, ko ogrodje IaaS ni arhitekturno zasnovano za podporo takšni povezljivosti. Na primer, izvorna koda ogrodja Eucalyptus vključuje lupinske skripte

Tabela 3.2

Primerjava platform IaaS z uporabo taksonomije in arhitekturnega ogrodja IaaS (prvi del).

Arhitekturno ogrodje IaaS	***OpenNebula***	***Eucalyptus***	***OpenStack***	***Nimbus***	***AWS***	***Microsoft Private Cloud***	***VMware vCloud***
Sloj abstrakcije virov							
• Compute	Oned	Internal	Nova-compute	Workspace Service	Amazon EC2	Microsoft Hyper-V Server	vCenter Server
• Storage	Internal	Walrus	Object Store (Swift)	Cumulus	Amazon S3	Ne podpira splošno-namenske shrambe oblaka	Ne podpira splošno-namenske shrambe oblaka
• Volume	Internal	Storage Controller	Nova-volume	Internal	Amazon EBS	Internal	vStorage Virtual Machine File System
• Omrežje	Virtual Network Manager	Internal	Nova-network	Workspace Control	Internal	Internal	vCloud Networking
Sloj jedrnih storitev							
• Identiteta	Auth Subsystem	/	Keystone	/	Amazon IAM	Active Directory Federation Services (AD FS)	vShield Manager
• Razporejanje	Scheduler	Cluster Controller	Nova-scheduler	Internal	ELB	Internaal	Distributed Resource Scheduling (DRS)
• Repozitorij slik	Image Repository	Image Management	Glance, nova-objectstore	Science Clouds marketplace	AMI repository	VMM Library	Catalogs, VMware virtual appliance marketplace
• Zaračunavanje	/	/	/	/	Internal	Service Manager Self-Service Portal	vCenter Chargeback
• Logiranje	Internal	Internal	Internal	Internal	Internal	VMM	Internal
Podporni sloj							
• Sporočilno vodilo	RabbitMQ	RabbitMQ	RabbitMQ	Internal	Amazon SQS	Internal	ActiveMQ
• Podatkovna baza	sqlite3, MySQL, PostgreSQL	sqlite3, HSQLDB	sqlite3, MySQL in PostgreSQL	Internal	Internal	MS SQL	Oracle DB in MS SQL
• Storitev za prenos	Transfer Manager	Node Controller	nova-objectstore	Workspace Service	Internal	Internal	Image Transfer
Sloj upravljanja							
• Upravljanje z viri	Internal	Internal	Internal	Workspace Control	AWS Management Console	System Center Virtual Machine Manager (VMM)	vCenter Server
• Upravljanje s federacijo	/	/	/	/	/	Znotraj postavitev Hyper-V, z uporabo Active Directory Federation Services (AD FS)	Znotraj postavitev vCloud, z uporabo vCloud Connector, vCenter Orchestrator
• Upravljanje elastičnosti	/	/	/	/	Auto Scaling	System Center Orchestrator	vCenter Orchestrator

Tabela 3.2

Primerjava platform IaaS z uporabo taksonomije in arhitekturnega ogrodja IaaS (drugi del).

Arhitekturno ogrodje IaaS	***OpenNebula***	***Eucalyptus***	***OpenStack***	***Nimbus***	***AWS***	***Microsoft Private Cloud***	***VMware vCloud***
• Upravljanje uporabnikov/skupin	Internal	Internal	Internal	Internal	IAM, AWS MFA	VMM	vCloud Director
• SLA definicija	Ogrodje bi moralo biti prilagojeno za delo z obstoječimi komponentami SLA	Ogrodje bi moralo biti prilagojeno za delo z obstoječimi komponentami SLA	Ogrodje bi moralo biti prilagojeno za delo z obstoječimi komponentami SLA	Ogrodje bi moralo biti prilagojeno za delo z obstoječimi komponentami SLA	Internal	Internal	vCloud Service Definition
• Spremljanje	Monitoring Subsystem	Možna integracija z zunanjimi sistemi	Možna integracija z zunanjimi sistemi	Možna integracija z zunanjimi sistemi	CloudWatch	System Center Operations Manager	vCloud Infrastructure Management
• Poročanje	/	/	/	/	AWS Management Console	Reporting Dashboard	vCenter Chargeback
• Upravljanje z incidenti	/	/	/	/	/	System Center Operations Manager	vCloud Operations Management (Incident and Problem Management)
• Upravljanje z energijo	/	/	/	/	/	System Center Configuration Manager	Distributed Power Management (DPM)
• Upravljanje z najemi	Možna je integracija s Haizea (open-source VM-based lease manager)	/	/	/	/	Service Manager Self-Service Portal	vCloud Director
Upravljavska orodja							
• CLI orodja	OpenNebula CLI 1.2	Euca2ools	Euca2ools, VNC Console	Cloud Client	Dostop do EC2 in S3	Powershell	vSphere Command-Line
• API-ji	OpenNebula Cloud API, EC2 Query API, The OpenNebula OCCI API	EC2 API, S3 API, Eucalyptus API	OpenStack API, EC2 API, S3 API, Swift API, Glence API	EC2 API, Nimbus WSRF	APIs provided for most AWS services	Hyper-V WMI	vCloud API
• Nadzorna plošča	SunStone, ElasticFox	ElasticFox, Admin UI	OpenStack Dashboard, Horizon	/	AWS Management Console	VMM SSP, Service Manager Self-Service Portal, App Controller	vCloud Director
• Orkestrator	/	/	/	/	/	System Center Orchestrator	vCenter Orchestrator
Sloj varnosti							
• Avtentikacija	Auth Subsystem	X509 certificate, SSH keys	X509 certificate, SSH keys, LDAP authentication	X509 certificate, SSL connection, Integrate Globus (certification)	AWS Security Token Service (STS), IAM, AWS MFA	Integrated Windows authentication (NTLM or Kerberos)	vShield Manager
• Avtorizacija	Auth Subsystem, ACL authorization system	Eucalyptus Identity Authorization and Management	Internal	Internal	IAM	Authorization Manager	vShield Manager

Tabela 3.2

Primerjava platform IaaS z uporabo taksonomije in arhitekturnega ogrodja IaaS (tretji del).

Arhitekturno ogrodje IaaS	***OpenNebula***	***Eucalyptus***	***OpenStack***	***Nimbus***	***AWS***	***Microsoft Private Cloud***	***VMware vCloud***
• Varnostne skupine	Internal	Internal	Internal	Internal	Internal	Domain Isolation, Group Policy	vShield Edge
• Enotna prijava (SSO)	/	/	/	/	IAM (federated identity)	Active Directory Federation Services (AD FS)	vCloud Connector (SSO)
• Spremljanje varnosti	/	/	/	/	AWS Management Console	VMM	vShield Edge
Nadzorniški sloj							
• Izvrševanje SLA-jev	/	/	/	/	Internal	System Center Operations Manager	vCloud Service Management
• Spremljanje SLA-jev	/	/	/	/	Internal	System Center Operations Manager	vCloud Service Management
• Merjenje	/	/	Telemetry Service (Ceilometer)	/	Internal	VMM	vCenter Chargeback
• Nadzor nad politikami	/	/	/	/	AWS Management Console	VMM, Service Manager	vCenter Configuration Manager
• Storitev za obveščanje	/	/	/	/	Amazon Simple Notification Service (SNS)	Internal	Internal
• Orkestracija	/	/	Orchestration Service (Heat)	/	/	System Center Orchestrator	vCenter Orchestrator
Storitve z dodano vrednostjo							
• Območja razpoložljivosti	/	/	/	/	Multi-AZ	Internal	vCloud resource groups
• Gruča (HA)	/	/	/	/	Internal	Cluster shared volumes (CSV)	vSphere High Availability
• Podpora hibridnemu oblaku	Arhitektura, ki temelji na adapterjih, omogoča komunikacijo s številnimi zunanjimi oblaki (Amazon EC2 and ElasticHosts)	/	/	Vključuje "EC2 backend", ki lahko posreduje zahteve k EC2	/	Deloma, z uporabo App Controller (zgolj znotraj oblakov podprtih s Hyper-V)	Deloma, z uporabo vCloud Connector (zgolj znotraj oblakov podprtih z vSphere)
• Migracija v času izvajanja	/	/	/	/	/	Live Migration	vMotion
• Podpora za portabilnost	/	/	/	/	/	/	/
• Kontekstualizacija slik	/	/	/	Nimbus Context Broker	/	/	/
• Podpora virtualnim namenskim napravam	/	/	/	/	/	Server Application Virtualization (Server App-V)	vApp

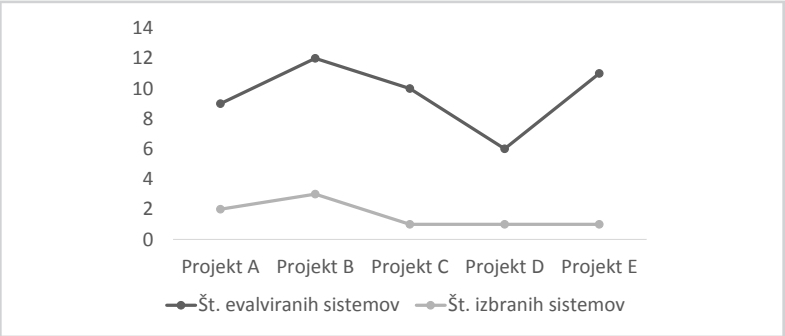
Tabela 3.3

Evalvacija taksonomije in arhitekturnega ogrodja na realnih projektih.

	Ime projekta	*Opis projekta*	*Št. evalviranih sistemov*	*Št. izbranih sistemov*	*Obdobje evalvacije*	*Stopnja uspeha*
Projekt A	KC OpComm	Kratkoročni cilj OpComm je oblikovati in graditi prototip odprte komunikacijske platforme, ki bo služil kot osnova za razvoj različnih novih naprednih storitev in aplikacij za internet naprav (angl. Internet of Things – IoT). Takšna platforma bo temeljila na najustreznejšem sistemu IaaS.	9 (3 komercialni, 6 odprto-kodnih)	2	2 meseca	18.2 %
Projekt B	KC Class	Glavni cilj je razvoj storitev in izdelkov na področju računalništva v oblaku (IaaS, PaaS in SaaS). Raziskave in razvoj sta osredotočena na naslednje teme: varnost, mobilnost, večnajemništvo, upravljanje identitete, varnost, upravljanje s podatki in odprti vmesniki.	12 (2 komercialna, 10 odprto-kodnih)	3	3.5 mesecev	19.4 %
Projekt C	SLA@SOI	Tehnični cilj projekta SLA@SOI je določiti celosten pogled za upravljanje s SLA-ji in implementirati ogrodje za upravljanje SLA-jev, ki ga je mogoče enostavno vključiti v storitveno usmerjeno infrastrukturo (angl. Service-Oriented Infrastructure – SOI), ki so razporejene na določenem sistemu v oblaku.	10 (3 komercialni, 7 odprto-kodnih)	1	3 mesece	7.7 %
Projekt D	Contrail	Glavni prispevek projekta Contrail je integriran pristop k virtualizaciji, sistemom IaaS, storitvam za federacijo oblakov IaaS ter sistemom PaaS. Njegov cilj je poenotenje in združitev sedanjih komercialnih oblakov v t.i. federirano računalništvo v oblaku.	6 (2 komercialna, 4 odprto-kodni)	1	1.5 mesecev	13.3 %
Projekt E	Telekom	Svetovalni projekt računalništva v oblaku.	11 (5 komercialnih, 6 odprto-kodnih)	1	1.5 mesecev	8 %

(angl. Shell Scripts), ki modificirajo konfiguracijske datoteke orodij Nagios in Ganglia, da bi omogočili Eucalyptus-specifično spremljanje na vnaprej definiranem številu gostiteljev. Večina ponudnikov javnih oblakov (npr. Amazon EC2 z uporabo CloudWatch) ponuja funkcionalnosti za dosedanje samodejne elastičnosti, kot odziv na povečanje oz. pomanjšanje obremenitev in temelji na vgrajenih sistemih za spremljanje. Vendar pa to ne drži za odprto-kodne sisteme IaaS in predstavlja številne priložnosti za implementacijo razširitev za samodejno skaliranje virov računalniškega oblaka.

V primeru dostave virov oblaka h končnim uporabnikom preko poslovnega modela ”plačaj glede na uporabo”, mora biti ponujenih večino mehanizmov omenjenih v prejšnjih paragrafih (npr. merjenje, zaračunavanje in spremljanje). Pravzaprav je ta model običajno eden izmed ključnih dejavnikov pri adaptaciji računalništva v oblaku s strani organizacij, saj predstavlja enega izmed razlogov za izrazito redukcijo stroškov. Nasprotno, komercialne rešitve podpirajo večino teh funkcionalnosti, vendar kljub temu ne dosegajo celotnega obeta računalniških oblakov. Manjka na primer podpora za portabilnost, naprednejša kontekstualizacija slik, poleg tega ponujajo podporo hibridnim oblakom ter upravljanju s federacijo zgolj znotraj lastniških (angl. Proprietary) namestitvah. VMwareova rešitev na primer ponuja hibridno podporo z uporabo tehnologije vCloud Connector, ki je izvedljiva zgolj znotraj vSphere-podprtih oblakov. Ob analizi in primerjavi različnih arhitektur IaaS identificiramo nekatere ključne pomanjkljivosti,



Slika 3.6

Število evalviranih sistemov proti številu izbranih sistemov.

kot tudi priložnosti za prihodnji razvoj in raziskave. Da bi lahko izpolnili vizijo računalništva v oblaku, bodo v prihodnje morale biti naslovljene omenjene pomanjkljive in nepodprte funkcionalnosti trenutnih arhitektur.

Predlagana taksonomija in arhitekturno ogrodje IaaS sta bila evalvirana na številnih realnih projektih, vključno z KC OpComm[1], KC Class[2], SLA@SOI[3], Contrail[4] ter na enem svetovalnem projektu za največjega telekomunikacijskega ponudnika[5] v Sloveniji. Tabela 3.3 vključuje pet projektov, pri čemer vsak ponuja kratek opis, število evalviranih sistemov IaaS, število izbranih sistemov, obdobje evalvacije (angl. Evaluation Period) ter stopnjo uspeha (angl. Success Rate). Stopnjo uspeha izračunamo, kot razmerje med številom izbranih sistemov ter vsote števila evalviranih sistemov in obdobja evalvacije (izračunanih v mesecih), skladno s spodnjo enačbo (enačba 3.1):

$$\text{stopnja uspeha} = \frac{\text{število izbranih sistemov}}{\text{št.evalviranih sistemov + obdobje evalvacije}} \tag{3.1}$$

V kontekstu odločanja znotraj organizacij IT, je bil projekt B smatran kot najbolj uspešen in projekt C, kot najmanj uspešen. Pri vsakem projektu je bil izbran vsaj en ustrezen infrastrukturni oblak (slika 3.6), kar kaže na uporabnost modela v realnem svetu ter na učinkovito validacijo predlagane taksonomije ter arhitekturnega ogrodja IaaS.

[1] http://www.opcomm.eu/en/
[2] http://www.kc-class.eu/
[3] http://sla-at-soi.eu/
[4] http://contrail-project.eu/
[5] http://www.telekom.si/

"Learn the rules like a pro, so you can break them like an artist."

— Pablo Picasso

4

Obogatitev sodobnih izvajalnih platform s specifikami oblaka

4.1 Uvod

Glede na analitsko hišo Gartner je sloj PaaS, ki je na večini diagramov prikazan kot vmesni sloj med SaaS in IaaS, definiran kot široka kolekcija storitev aplikacijske infrastrukture (tj. vmesna programska oprema), kar vključuje platformo aplikacije, integracijo, upravljanje s poslovnimi procesi ter storitvami podatkovne baze [154]. Kljub temu se večina obstoječih poslovnih aplikacij še vedno izvaja na platformah (npr. Java EE in .NET), ki niso popolnoma primerne za izvajanje v okolju oblaka, saj niso bile zasnovane z upoštevanjem specifik računalniškega oblaka. Računalništvo v oblaku bo v prihodnje potrebovalo specializirane tehnologije in arhitekture, ki so izvorno načrtovane za okolja oblakov [155]. Ker bodo platforme, ki temeljijo na vsebnikih, predstavljale platformo izbire pri številnih oblakih PaaS, bodo morali vpeljati številne spremembe, da bi dosegli pravo obljubo računalniških oblakov. Elastičnost, večnajemništvo in nadzor oblaka so trije najpomembnejši aspekti, ki morajo biti naslovljeni, da bi brezhibno izvajali aplikacije v oblakih IaaS in PaaS. Čeprav je to področje relativno mlado, je bilo izvedenih že precej raziskav in študij [46, 47, 155], ki povezujejo paradigmo računalništva v oblaku s sodobnimi platformskimi vsebniki. Kljub temu omenjena dela naslavljajo zgolj večnajemniške karakteristike in ne vpeljujejo elastičnost ter nadzora oblaka v te platforme (na višji nivo abstrakcije), saj je interakcija z oblakom v večinski meri izvedena preko nizkonivojskih API-jev in vmesnikov z ukazno vrstico [156]. Vpeljava omenjene napredne funkcionalnosti aplikacijam omogoča (1) koristiti vse potenciale oblakov IaaS in PaaS (npr. dinamično skaliranje aplikacij z namenom zadovoljitve visokih obremenitev - tako ročno, kot tudi samodejno - dinamična alokacija virov ter samo-oskrbovanje virov; izvajanja različnih delov aplikacij v priporočljivih podatkovnih centrih, da bi zadostili performančnim in regulatornim predpisom; ali zmožnost pridobivanja metrik za spremljanje) in (2) ponuditi aplikacijskim administratorjem in programerjem večji nadzor (kompleksnost spodaj ležečih infrastrukturnih sistemov je pri tem prikrita).

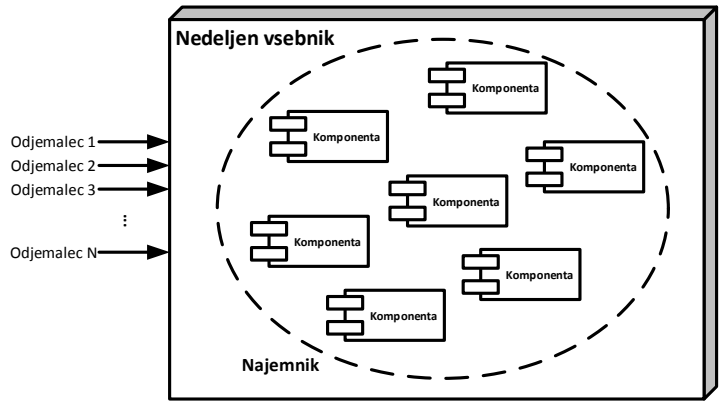
Ključna kontribucija četrtega poglavja je definirati generičen sistem parametrov za izvajanje aplikacij v oblakih IaaS in PaaS. Takšen formalni sistem z uporabo politik in programskih direktiv transformira aplikacijske zahteve, ki jih izraža administrator aplikacije ali razvijalec, v operacije nad spodaj ležečimi viri IaaS. S predlaganim pristopom naslavljamo vse deficitne funkcionalnosti (tj. elastičnost in nadzor oblaka) obstoječih platform, ki temeljijo na vsebnikih, in jih vpeljujemo na nivo aplikacije in platforme.

Skladno s tem strukturiramo sistem parametrov v dve skupini parametrov: parametri za nadzor in parametri elastičnosti. Parametri za nadzor vpeljujejo tri parametre: (1) parameter za spremljanje aplikacije (angl. Application Monitoring Parameter), (2) parameter za konfiguracijo lokacije (angl. Application Location Configurations Parameter) ter (3) parameter za kršitve SLA-jev (angl. SLA Violation Parameter). Skupina parametrov elastičnosti vpeljuje (1) prameter za skaliranje poslovnega sloja (2) analogen parameter za spletni sloj ter (3) parameter za definiranje uporabniških pravila za elastičnost (angl. User-Defined Elasticity Rules), ki temeljijo na metrikah relevantnih za celotno aplikacijo. Parameter za definiranje uporabniških pravila za elastičnost vpeljuje pristop, ki pri avtonomnem skaliranju kombinira stroške, kakovost in delovne obremenitve. Obe skupini parametrov predstavljata platformsko-neodvisne parametre, ki jih uporabljamo za preslikavo v platformsko-specifične programske direktive in politike (v našem primeru so parametri preslikani v Java EE Annotation Metadata). Preslikava med sistemom parametrov in sistemom metapodatkov Java EE je bila realizirana v obliki deklaracij novih anotacijskih tipov ter razširitev postavitvenih deskriptorjev.

4.2 Pregled obstoječih izvajalnih platform

Vsebniki (angl. Containers) so izvajalna okolja, ki komponentam ponujajo določene storitve, kot so na primer upravljanje z življenjskim ciklom (angl. Life-Cycle Management), podpora transakcijam (angl. Transaction Support) in storitve varnosti (angl. Security Services) [157]. Pravzaprav lahko vsak tip platforme definira različne življenjske cikle, storitve ter API-je za gostujoče komponente [48]. Kar pomeni, da lahko različni oblaki PaaS temeljijo na različnih platformah, na primer Google App Engine (GAE) [158] ponuja vsebnike za servlete, skripte za Python ter programski jezik Go, Windows Azure [159] ponuja okolje za mobile, .NET, Node.js, PHP, Java in Python aplikacije, medtem ko AWS Elastic Beanstalk [160] ponuja platformo za .NET, PHP, Python, Ruby in Java razvijalce. Čeprav se te rešitve razlikujejo v aspektih, kot so podprti programski jeziki, aplikacijska ogrodja itn., obstajajo inherentne podobnosti pri načinu kako ti sistemi upravljajo življenjski cikel aplikacij, ki so postavljene na teh okoljih [161]. To predstavlja priložnost definiranja generičnega sistema parametrov, ki naslavlja vse pomanjkljive zmogljivosti sodobnih vsebniških platform (angl. Container-Based Platforms) in jih aplicira na nivo aplikacij in platforme z uporabo programskih direktiv (angl. Programming Directives) in politik (angl. Policies).

Oblaki PaaS se obravnavajo kot moderne platforme, ki temeljijo na vmesnikih, in

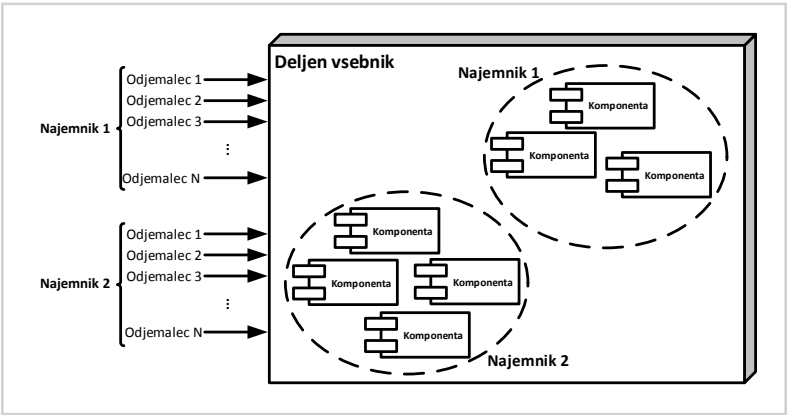


Slika 4.1

Konceptualna zasnova nedeljenega vsebnika.

so primerne za izvajanje v okolju oblaka. Takšni sistemi ponujajo okolje, ki temelji na vsebnikih, in kjer lahko uporabniki postavljajo svoje aplikacije v obliki komponent programske opreme (angl. Software Components) ter ponujajo nabore knjižnic in storitev za podporo načrtovanja, razvoja, testiranja, postavitve in vzdrževanja aplikacij v oblaku [48]. Najpomembnejša komponenta arhitekture PaaS je izvajalno okolje, ki mora izpolniti tipične zahteve, kot so skalabilnost, nadzor, zanesljivost in varnost. Izvajalna okolja, ki temeljijo na vsebnikih v večini primerov podpirajo večnajemniško arhitekturo (angl. Multi-Tenancy Architecture), ki omogoča deljenje/izmenjavo ene instance aplikacije med več najemniki (angl. Tenants) oz. uporabniki. Kljub manjši fleksibilnosti v primerjavi z oblaki IaaS, oblaki PaaS postajajo vedno bolj pomembne platforme za hitrejšo izgradnjo aplikacij računalniških oblakov (angl. Cloud-Based Applications). Oblaki IaaS zahtevajo od razvijalcev in sistemskih administratorjev, da nameščajo in konfigurirajo celoten sklad programske opreme (angl. Software Stack), ki ga aplikacije potrebujejo, medtem ko oblaki PaaS ponujajo vnaprej pripravljeno izvajalno okolje ter storitve za aplikacije. Glede na nivo izolacije (angl. Isolation Level) platformskih vsebnikov, lahko oblake PaaS klasificiramo v dve skupini [46]:

- Oblaki, ki zgradijo unikatno okolje za posameznega najemnika, pri čemer je to okolje nameščeno na nedeljene (angl. Non-Shared) virtualne stroje. To pomeni, da sistemi PaaS postavljajo komponente vsakega najemnika na različne virtual-



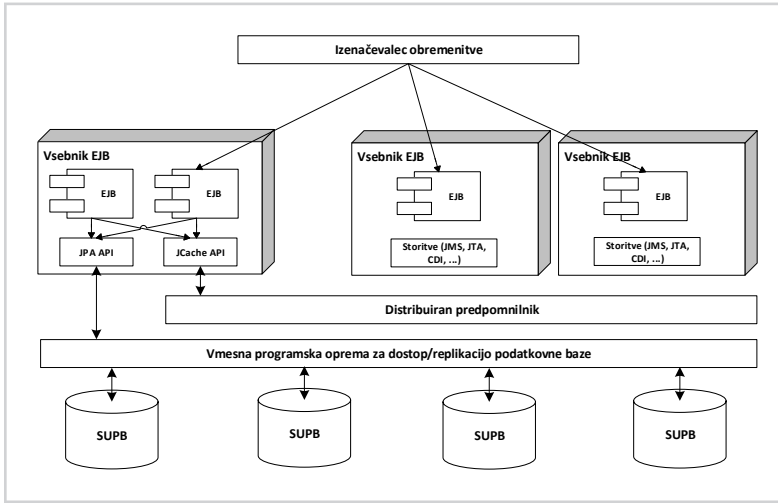
Slika 4.2

Konceptualna zasnova deljenega vsebnika.

ne/ločene stroje. V teh sistemih je ponudnik virtualnih strojev (tj. ponudnik IaaS) zadolžen za implementacijo primerne stopnje izolacije (slika 4.1).

- Oblaki, kjer so komponente od različnih najemnikov postavljene na iste sisteme vsebnikov (angl. Container Systems), tako da lahko ponudnik gosti več aplikacij v istem bazenu virov. Da bi dosegli varen večnajemniški model v platformah PaaS, se mora vsaka aplikacija izvajati izolirano od drugih, tako da zlonamerne aplikacije in aplikacije v okvari ne morejo vplivati na druge aplikacije. Iz tega razloga to doktorsko delo ne naslavlja takšnih sistemov, saj zahtevajo ponovno načrtovanje izvajalnih okolij (vsebnikov) trenutnih platform (slika 4.2).

Upravitelji virtualne infrastrukture, kot so Eucalyptus, OpenStack, Microsoft Private Cloud, VMware vCloud itn., so zadolženi za vrsto opravil, ki jih izvajajo v privatnem okolju oblaka IaaS. Zadolženi so za upravljanje z virtualnimi stroji/gostitelji/gručami, razporejanje ter izenačevanje obremenitev med različnimi virtualnimi stroji, elastično razširjanje kapacitet v javne oblake itn. Zagotavljanje elastičnosti je torej ena izmed ključnih elementov vsakega upravitelja virtualne infrastrukture. Le-ti zagotavljajo elastičnost s podpiranjem horizontalne skalabilnosti virtualnih strojev, ki je lahko dosežena ročno ali samodejno. Na nivoju PaaS mora biti skalabilnost obravnavana na dveh temeljnih elementih platforme kot storitve: vsebnikih in podatkovni bazi. Horizontalna skalabilnost je v obeh primerih zagotovljena preko mehanizma replikacije.



Slika 4.3

Arhitektura večslojne aplikacije v okolju oblaka.

Arhitektura takšne večslojne aplikacije na primeru platforme Java EE je prikazana na sliki 4.3.

4.3 Predlagan sistem parametrov

Glede na literarni pregled najpomembnejših komercialnih in odprto-kodnih platform PaaS [2, 162], študiji trenutnih in prihodnjih tehnoloških trendov paradigme PaaS [163, 164], ter testiranje posameznega sistema PaaS v številnih realnih projektih, identificiramo nekatere slabosti in pomanjkljivosti sodobnih platform, ki temeljijo na vsebnikih.

Trenutne raziskave oblaka dajejo velik poudarek nizkonivojskim tehnološkim usmeritvam in ne naslavljajo zmožnosti upravljanja življenjskega cikla storitev oblaka na višjih nivojih abstrakcije. Na primer, da bi samodejno povečali/zmanjšali količino uporabljenih virov skladno z varianco delovnih obremenitev. Večina znanstvenih del implementira pravila za skaliranje zgolj z uporabo infrastrukturnih metrik in ne naslavljajo metrike, ki so relevantne za aplikacijo. Posledično so številni avtorji ponudili načine za opis storitev oblaka (tj. aplikacij) na holističen način, medtem ko so ponudili primeren nivo abstrakcije in omogočili definiranje akcij za samodejno skaliranje, ki temeljijo na poljubnih metrikah storitve oblaka [22]. Kljub temu je takšno upravljanje storitev

oblaka (od začetka do konca) prisotno zgolj na nivoju IaaS (v večini primerov z uporabo formata OVF) in ne vpeljuje zmožnosti na višjih nivojih abstrakcije (tj. aplikacija ali platforma). Ker je samodejno oskrbovanje in upravljanje storitev oblaka eden ključnih zahtev računalništva v oblaku, predstavlja celostno upravljanje življenjskega cikla storitev oblaka (npr. skaliranje, spremljanje, nadgrajevanje, namestitev popravkov itn.) na nivoju platforme in aplikacij velik izziv, ki ga je potrebno nujno nasloviti. Poleg tega lahko izvajanje aplikacij na elastičen način, ki upoštevajo vnaprej definirane preference in omejitve uporabnika, predstavlja velik izziv. Medtem ko je prisotnih veliko sorodnih raziskav na področju adaptivnega oskrbovanja storitev (angl. Adaptive Service Provisioning), se običajno ne upoštevajo kombinacije virov, stroškov in kakovosti, kot trije najpomembnejši aspekti elastičnosti oblaka [165]. Z upoštevanjem teh dejstev, nekatere platforme oblaka ponujajo vmesnike in API-je, ki omogočajo uporabnikom ročno spreminjanje virov (tj. ročna elastičnost), medtem ko druge platforme ponujajo rešitve, ki vključujejo polno avtomatizirane storitve za spremljanje in samodejno alokacijo virov (tj. samodejna elastičnost). Kljub temu trenutno ni prisoten sistem, ki bi vključeval oba pristopa in jih z uporabo politik in programskih direktiv vpeljeval na nivoju platforme in aplikacije. Takšna izboljšava bi ponudila aplikacijskim administratorjem in programerjem večji nadzor in možnost lažjega upravljanja, pri čemer bi jim omogočala definicijo politik elastičnosti (angl. Elasticity Policies) na intuitiven način in pri tem skrila kompleksnost spodaj ležečih infrastrukturnih sistemov.

Še več, trenutni sistemi PaaS so omejeni s ponujenimi zmožnostmi ponudnika in ne omogočajo enostavne razširljivosti ali kostumizacije s strani uporabnika (npr. administratorja aplikacije) oz. razvijalca [166], saj ne obstajajo politike in programske direktive, specifične za okolje oblaka. Iz tega razloga spodaj ležeča vmesna programska oprema PaaS in aplikacije, ki so na njej zgrajene, niso portabilne med različnimi javnimi/privatnimi oblaki in regijami. Različni deli več nivojskih aplikacij (tj. aplikacijski sloji) ne morejo biti postavljeni na geografsko ločenih regijah oz. podatkovnih centrih, kar bi prišlo prav v številnih scenarijih. Na primer sloj podatkovne baze (angl. Database Tier) želimo postaviti v lokalni podatkovni center, da bi zadovoljili skladnost z lokalnimi regulativami (angl. Local Regulatory Compliance), medtem ko želimo spletni sloj (angl. Web Tier) postaviti na strateško določene lokacije (bližje končnim uporabnikom), da bi ponudili maksimalno pasovno širino ter nižji odzivni čas za dostavo spletne vsebine. Pravzaprav trenutni programski pristopi, ki so namenjeni okolju oblaka, niso dovolj fleksibilni in učinkoviti pri podpiranju generičnih aplikacijah, ki bi jih

lahko simultano postavili med infrastrukture različnih ponudnikov [156]. Propagacija teh zmožnosti na višji nivo abstrakcije, bližje razvijalcem in aplikacijskim administratorjem, bi omogočila večji nadzor in lažje upravljanje aplikacij, ki se izvajajo v okolju oblaka.

Spremljanje oblaka se običajno navezuje na spremljanje performance fizičnih in virtualnih virov, vmesne programske opreme in aplikacij, ki se izvajajo nad njimi. Na podoben način številna znanstvena dela dajejo velik poudarek na spremljanje uporabniške izkušnje (tj. kakovosti) in stroškov [167], ki sta ob spremljanju računalniških virov in aplikacije, dva najpomembnejša aspekta spremljanja v okolju oblaka. Kljub prisotnosti številnih orodij za spremljanje oblakov, ki so sposobni spremljati številne metrike virov (angl. Resource Metrics), aplikacijske metrike (angl. Application-Specific Metrics), stroškovne metrike (angl. Cost Metrics) ter metrike kakovosti (angl. Quality Metrics), je zmožnost pridobivanja teh relevantnih metrik na nivoju aplikacije, z uporabo programskih direktiv in politik, še zmeraj odprt problem. Pri tem je potrebno vedeti, da prej omenjene metrike pokrivajo vse nivoje sklada oblaka, vključno s poslovnimi metrikami. Poleg tega zagovarjamo stališče, da predstavlja zmožnost odzivanja (na primer sprožitev notifikacije) aplikacije na zaznane kršitve SLA, pomembno dodano vrednost razvijalcem programske opreme, in si zasluži dodatno pozornost tako v znanstvenem svetu, kot tudi v industriji. Skratka, različne tehnike in storitve spremljanja lahko pripomorejo razvijalcu aplikacij v smislu zagotavljanja visoke učinkovitosti izvajanja storitev oblaka; detektiranje variacij pri performanci storitev oblaka; ter obračunavanje kršitve SLA za določene parametre QoS [168–170].

Za uspešno realizacijo prej omenjene vizije moramo nasloviti številne ključne izzive in jih aplicirati na raznolike platforme, da bi dosegli obogatitev s pomanjkljivimi zmožnostmi elastičnosti in sistemskega nadzora, ki so specifične za okolje oblaka. Iz tega razloga, je naš cilj zasnovati generičen sistem parametrov (angl. Parameter System), ki preko politik in programskih direktiv transformira aplikacijske zahteve, izražene s strani aplikacijskih administratorjev ali razvijalcev, v operacije nad spodaj ležečo infrastrukturo oblaka (npr. oskrbovanje VM-ja). V tem primeru se lahko aplikacijski administrator/razvijalec osredotoči na aplikacijo in njeno postavitveno okolje, ne da bi pri tem moral biti še v vlogi sistemskega administratorja (angl. System Administrator), administratorja podatkovne baze in administratorja vmesne programske opreme (kot je to primer pri IaaS) [161]. Skladno s tem strukturiramo sistem parametrov v dve skupini parametrov: parametri za nadzor (angl. Control Parameters) in parametri

elastičnosti (angl. Elasticity Parameters). Predlagan pristop predstavlja ključni diferenciator generičnega sistema parametrov v zvezi z obstoječimi pristopi iz akademskega sveta in industrije.

4.3.1 Parametri za nadzor

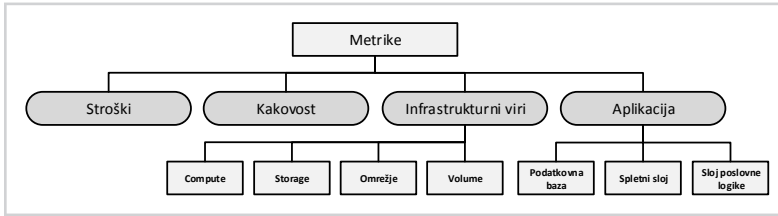
Skupina parametrov za nadzor naslavlja nadzorniški aspekt aplikacij, ki se izvajajo v okolju oblaka. Ti parametri vključujejo:

- *Parameter za spremljanje aplikacije* – zmožen pridobiti metrike infrastrukturnih virov, aplikacijske, stroškovne metrike ter metrike kakovosti/uporabniške izkušnje v oblaku.
- *Parameter za konfiguracijo lokacije* – zadolžen za specifikacijo regije/podregije, v kateri se aplikacija izvaja, z namenom zadovoljitve zahtev po performanci oz. skladnosti s predpisi.
- *Parameter za kršitve SLA-jev* – omogoča definicijo mehanizma povratnega klica za signalizacijo kršitve dogovora SLA, pri čemer je lahko odziv/signal karkoli, od proženja notifikacije preko e-pošte, klicanja določene operacije skaliranja, vse do izvajanja poljubnega orkestracijskega delovnega toka.

Parameter za spremljanje aplikacije

Čeprav so trenutna orodja za spremljanje oblakov zmožna slediti številnim infrastrukturnim metrikam, aplikacijskim metrikam, stroškovnim metrikam in metrikam kakovosti (tj. uporabniške izkušnje), je zmožnost pridobivanja teh metrik na aplikacijskem nivoju, z uporabo programskih direktiv in politik, še vedno velika pomanjkljivost trenutnih sistemov oblaka. Takšna obogatitev sistemov bi lahko bila uporabljena za:

1. programsko proženje politik z namenom avtomatizacije kompleksnih upravljavskih opravil znotraj okolja oblaka,
2. uporabo pridobljenih metrik za demonstracijo realno-časovnega stanja določenih storitev oblaka (npr. performanca in stanje storitve) ali celotne infrastrukture oblaka.



Slika 4.4

Kategorizacija metrik za spremljanje.

Distribuirana narava računalništva v oblaku omogoča postavitev dolgotrajnih poizvedb v omrežju, z namenom spremljanja raznolike množice entitet in metrik, ki se razširjajo čez celoten sklad oblaka [171]. Iz tega razloga, naša predlagana rešitev agregira metrike infrastrukturnih virov, aplikacijske metrike, stroškovne metrike in metrike kakovosti/uporabniške izkušnje, ter jih vpeljuje v obliki programskih direktiv, ki so uporabljene za injiciranje realno-časovnega stanja sistema v lokalne programske spremenljivke (angl. Variables). Te spremenljivke so lahko potem uporabljene za nadaljnjo uporabo znotraj programske kode.

V doktorskem delu zagovarjamo stališče, da kategorije, prikazane na sliki 4.4, predstavljajo holističen nabor metrik za spremljanje, ki so nujno potrebne pri doseganju vpeljave elastičnosti in nadzora oblaka. Prvi tip metrik so stroškovne metrike, ki so pridobljene s spremljanjem stroškov za uporabo virov pod različnimi pričakovanimi in dostavljenimi nivoji kakovosti. V našem primeru je poraba virov določena z uporabo merjenja virtualnih strojev v specifični časovni periodi, ki temelji na izkoriščenosti, in je definirana s strani administratorja aplikacije. Na primer določitev mesečne cene za instance računsko-intenzivnih (angl. Compute-Intensive) virtualnih strojev. Slednje je doseženo z merjenjem izkoriščenosti virov in kalkulacije ustreznih operativnih stroškov v določenem časovnem obdobju. Glede na T.T. Huu et al. [172] je celoten infrastrukturni strošek za izvajanje aplikacije izračunan kot vsota celotnih stroškov računske moči infrastrukture alocirane za aplikacijo (C_r) in celotnih stroškov za podatkovni prenos (C_b) (enačba 4.1):

$$C = C_r + C_b \tag{4.1}$$

V tem primeru je C_r izračunan kot (enačba 4.2):

$$C_r = c_r \times \sum_{i=1}^{s} m_i \times (Td_i + T_i(m_i, n, b)) \tag{4.2}$$

Pri tem c_r predstavlja stroške na sekundo za računske vire, s je število izvajajočih faz aplikacije, m_i je število vozlišč uporabljenih pri vseh fazah izvajanja, Td_i je čas postavitve (vključno s časom rezervacije virov in časom inicializacije) ter $T_i(m_i, n, b)$ predstavlja čas izvajanja v koraku i in je parametriziran s številom rezerviranih virov m_i, številom vhodnih podatkov za procesiranje n in pasovno širino b. Nadalje je C_b izračunan, kot:

$$C_b = c_b \times \sum_{i=1}^{s} (Td_i + T_i(m_i, n, b)) \sum_{j=1}^{k_i} b_j \tag{4.3}$$

Enačba 4.3 sešteje vse stroške podatkovnega prenosa, ki so vključeni pri izvajanju delovnega toka, vključno z vhodnimi podatki delovnega toka, ki je prenesen od zunaj (v koraku 1), z začasnimi podatki generiranimi med izvajanjem delovnega toka (v vseh korakih) ter izhodnimi podatki, prenesenimi do zunanjih virov (v koraku s). Pri tem c_b predstavlja ceno pasovne širine na en Mbps.

V tem kontekstu je dodatno zahtevan tudi fleksibilen sistem za merjenje, ki lahko upošteva izkoriščenost virov, ki si jih izmenjujejo različni najemniki [173]. Za ponazoritev naša predlagana rešitev primerja parametre (tj. metrike) s specifično maksimalno ceno, ki smo jo pripravljeni plačati za uro instance (t. i. cena ponudbe) in lahko proži te instance, dokler je *spremljana cena ≤ceni ponudbe* (angl. Bidding Price). To se zgodi, v primeru da instanca ni eksplicitno terminirana, ali se cena spremljane instance dvigne nad uporabnikovo ceno ponudbe. V primeru da aplikacijski administrator nenadoma zmanjša ceno ponudbe (tj. razpoložljiv mesečni proračun), se določeni sloji aplikacije migrirajo na cenejše instance, ki so skladne z zmanjšano ceno ponudbe. Podoben pristop je realiziran pri rešitvi Amazon EC2 Spot Instances [174].

Metrike kakovosti (tj. uporabniške izkušnje) služijo kot celovit indikator stanja storitve, pri čemer merijo kakovost uporabniške izkušnje, ki je merjena iz perspektive resničnih uporabnikov. Večina današnjih ponudnikov storitev (angl. Service Providers – SPs) se zanaša na neprekinjeno delovanje aplikacije (angl. Application Uptime), kot ključno metriko za evalvacijo performance aplikacije. Čeprav jim te metrike omogočajo merjenje razpoložljivosti storitve oblaka, ki so dostavljene končnim uporabnikom,

ponudniki storitev ne morejo meriti dejanske kakovosti storitve, ki jih izkusijo poslovni uporabniki. Zaradi tega obstaja potreba po uporabi performančnih metrik, kot sta odzivni čas aplikacije (angl. Application Response Time) in razpoložljivost aplikacije (angl. Application Availability), ki na tak način ponudnikom storitev omogočata spremljanje hitrosti aplikacij in evalvacijo kakovosti uporabniške izkušnje. Primer: metriko uporabniške izkušnje lahko uporabljamo za primerjavo časa nalaganja (angl. Load Time) specifičnega objekta, latenco strani, razpoložljivost storitve, ali sejo najemnika, s sprejemljivim časovnim intervalom specificiranim v obliki praga. Na primer *čas nalaganja* > 200ms, *latenca strani* > 75ms, *razpoložljivost* ≥99% ali *seja* > 10s. Glede na vnaprej definirana pravila preslikave (angl. Predefined Mapping Rules), shranjena v podatkovni bazi, so spremljane nizkonivojske infrastrukturne metrike, kot so *čas prekinitve* (angl. Down Time), *čas neprekinjenega delovanja* (angl. Uptime), *razpoložljiva shramba* (angl. Available Storage), periodično preslikane v visoko-nivojske metrike, kot so *razpoložljivost storitve* (angl. Service Availability). *Razpoložljivost storitve* Av lahko izračunamo z uporabo metrike *časa prekinitve* in metrike *časa neprekinjenega delovanja*, kot je definirano v enačbi preslikave (enačba 4.4):

$$A_v = (1 - \frac{\text{čas prekinitve}}{\text{čas neprekinjenega delovanja + čas prekinitve}}) * 100 \tag{4.4}$$

Naslednji primer je *odzivni čas* večslojne aplikacije (v našem primeru označen kot Rt), ki ga lahko izračunamo, ko vsoto odzivnih časov posameznega sloja storitve oblaka, ki ga označujemo kot $r(S_i)$ [137](enačba 4.5):

$$R_t = \sum_{i=1}^{m} r(S_i) \tag{4.5}$$

V enačbi zgoraj je razvidno, kako iz prvega sloja omrežja m-slojne aplikacije prispe nova zahteva končnega uporabnika. Potem ko so zahteve obdelane, so takoj posredovane naslednjim slojem in zapustijo aplikacijo pri končnemu sloju, ko so enkrat preiskale vse sloje m.

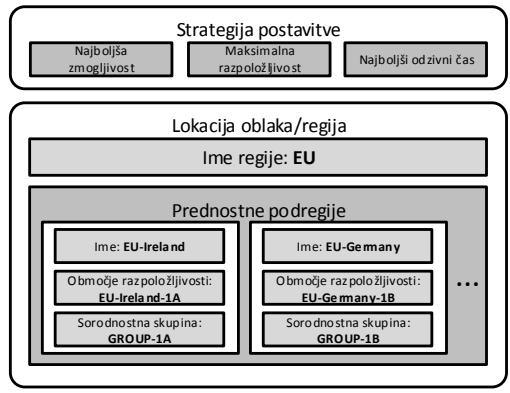
Infrastrukturne metrike so nizkonivojske metrike virov, ki so zadolžene za merjenje izkoriščenosti specifičnih infrastrukturnih virov, kot so računska moč (npr. izkoriščenost CPU-ja), shramba (npr. branje/pisanje na disk in razpoložljiva shramba), omrežje (npr. omrežje "in/out") in komponenta Volume (npr. celotni čas branja/pisanja na nosilec). Aplikacijske metrike vključujejo visoko-nivojske metrike, ki so izpeljane

iz storitve oblaka in vključujejo metrike za podatkovno bazo, metrike za spletni sloj ter metrike za poslovni sloj. Primer metrik za spletni in poslovni sloj je število aktivnih sej na spletni strežnik (angl. Active Sessions Per Web Server), število aktivnih sej na aplikacijski strežnik (angl. Active Sessions Per Application Server), število aktivnih uporabnikov, število aplikacijskih najemnikov, število instanc na določen aplikacijski sloj, število sporočilnih vrst itn. Metrike podatkovne baze so nadalje klasificirane v metrike relacijske podatkovne baze in metrike nerelacijske podatkovne baze (npr. NoSQL). Primer metrik relacijske baze je število uporabljenih povezav do baze, število razpoložljivega prostora za shrambo podatkovne baze itn. Po drugi strani pa metrike baze NoSQL vključujejo število uspešnih zahtev v specifični časovni periodi, število vrnjenih elementov s strani operacije scan/query itn. Vse aplikacijsko-specifične metrike so lahko zbrane z upoštevanjem določenih najemnikov znotraj aplikacije, specifične za okolje oblaka. To pomeni, da so lahko določene metrike pridobljene ločeno za posameznega najemnika aplikacije.

Parameter za lokacijsko konfiguracijo izvajanja aplikacij

Trenutne aplikacije, specifične za okolje oblaka, niso portabilne med različnimi javnimi/privatnimi oblaki in regijami, saj različni deli večslojne aplikacije ne morejo biti postavljeni v geografsko razpršenih regijah in podatkovnih centrih. Trdimo, da zmožnost prenosa lokacijske konfiguracije na višji nivo abstrakcije, ki je bližje razvijalcem in aplikacijskim administratorjem, omogoča lažji nadzor in enostavnejše upravljanje aplikacij postavljenih v okolju oblaka. Takšna obogatitev sodobnih izvajalnih platform lahko ponudi številne prednosti pred obstoječimi pristopi, ki niso dovolj učinkoviti in fleksibilni pri podpori večslojnim aplikacijam za simultano postavitev med več geografsko razpršenih oblakov. Iz tega razloga predlagamo razširitev v obliki politik, ki omogočajo specificiranje lokacije za postavitev in izvajanja za posamezne sloje storitve oblaka. Vsaka lokacija/regija oblaka vključuje ime regije in eno ali več prednostnih podregij (angl. Preferred Subregions). Poleg tega definiramo komplementarno komponento strategija postavitve (angl. Deployment Strategy). Strategija postavitve je uporabljana za identifikacijo ene izmed treh razpoložljivih strategij za postavitev posameznega sloja storitve oblaka. Te strategije so: *najboljša zmogljivost, maksimalna razpoložljivost* in *najboljši odzivni čas* (slika 4.5).

Strategija *najboljša zmogljivost* opisuje, da mora postavitvena platforma obdržati virtualne stroje čim bližje drug drugemu, na primer na istem fizičnem gostitelju ali znotraj



Slika 4.5

Strukturna arhitektura parametra za lokacijsko konfiguracijo izvajanja aplikacij.

hitre omrežne povezljivosti med gostitelji. Omogoča predvsem povezovanje virov znotraj istega podatkovnega centra. Primer: želeli bi obdržati podatke in izvorno kodo aplikacije na isti lokaciji, da bi zagotovili boljšo zmogljivost ali da bi dosegli skladnost z lokalno zakonodajo.

Strategija *maksimalna razpoložljivost* je zadolžena za postavitev instanc storitve oblaka v več območij razpoložljivosti (angl. Availability Zones), ki na tak način zaščiti aplikacijo pred napakami (prenehanjem delovanja) na eni sami lokaciji. Pomembno je, da se neodvisni sloji aplikacije izvajajo na enem ali več območjih razpoložljivosti, bodisi znotraj iste regije ali v drugi regiji, tako da lahko v primeru prenehanja delovanja enega območja, aplikacija v drugem območju deluje brez prekinitve. V ta namen mora ta strategija naslavljati:

1. migracijo delovnih obremenitev okolja aplikacije med različnimi regijami,
2. migracijo realno-časovne kopije podatkov med dvema ali več regijami (tj. podatkovna sinhronizacija),
3. zmožnost vzpostavitve omrežnega prometa med regijami.

Tretjo strategijo, *najboljši odzivni čas*, lahko uporabimo za dostavo dinamične, statične ali pretočne vsebine (angl. Streaming Content) aplikacij v oblaku, tako da uporabljamo globalno omrežje robnih lokacij (angl. Edge Locations). Na tak način so

zahteve po vsebini samodejno usmerjene k najbližji robni lokaciji, tako da je vsebina dostavljena z najboljšim možnim odzivnim časom. Prednostna podregija specificira priporočljive lokacije, kjer naj se izvaja določen del storitve oblaka in vsebuje ime prednostne regije, območje razpoložljivosti in sorodnostno skupino (angl. Affinity Group). Območje razpoložljivosti definira območje znotraj podatkovnega centra, medtem kot sorodnostna skupina specificira, katere vire znotraj iste regije je potrebno med seboj povezati. Sorodnostne skupine predstavljajo opcijski parameter, ki ustreza omogočeni strategiji *najboljša zmogljivost*, in je smiseln samo znotraj postavitev v istem podatkovnem centru oz. regiji.

Parameter za kršitve SLA

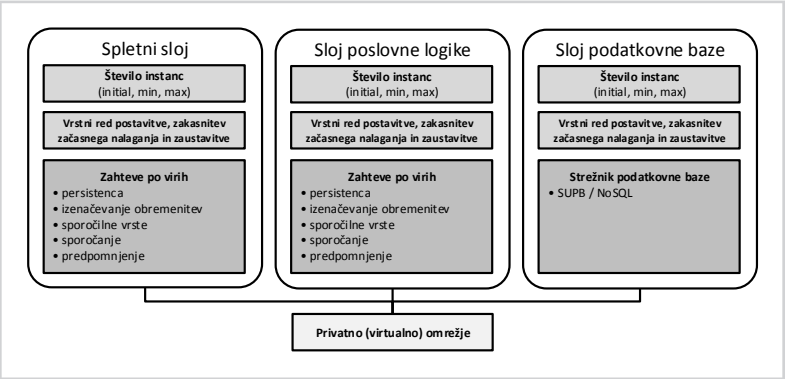
Bistvo uporabe SLA-jev v oblaku je zagotavljanje določenega nivoja kakovosti storitev končnim uporabnikom in vzpostavitev zagotovila, da bo ponudnik plačal kazen zaradi kršitve pogodbe v primeru, da dogovorjen nivo storitve ni izpolnjen. Zaradi dinamične narave oblaka, je za uspešno izvršitev SLA potrebno zagotoviti kontinuirano spremljanje atributov QoS, kot so čas izvajanja (angl. Execution Time), razpoložljivost in cena. Tehnike spremljanja pravzaprav pripomorejo k detektiranju variacij pri zmogljivosti storitve oblaka (tj. kršitev določenih atributov QoS). Skladno s tem je cilj predlagane rešitve detektiranje kršitev SLA preko spremljanja infrastrukturnih virov, aplikacije, stroškov/porabe ter uporabniške izkušnje računalniških oblakov. Detekcija možnih kršitev SLA temelji na vnaprej definiranih ciljih na ravni storitve, ki vsebujejo formalni izraz zagotovljenih pogojev storitve v danem obdobju [175, 176]. V omenjenem primeru se parametre QoS primerja s specificiranimi pragovi, ki pa lahko predstavljajo opozorilni prag (angl. Warning Threshold) ali prag kršitve (angl. Breach Threshold). Posledično parameter za kršitve SLA omogoča definicijo mehanizma povratnega klica (angl. Call Back) za signalizacijo kršitve dogovora SLA, pri čemer je lahko odziv/signal karkoli, od proženja notifikacije preko e-pošte, klicanja določene operacije skaliranja (tj. operacija scaleUp ali scaleDown), vse do izvajanja poljubnega orkestracijskega delovnega toka. Zaradi tega naš predlagan parameter vključuje programske direktive, ki jih lahko definiramo nad poljubno funkcijo povratnega klica. Takšna funkcija je zadolžena za detekcijo in ustrezno odzivanje na določene kršitve SLA. Kršitev SLA je posledica prekoračitve opozorilnega praga ali praga dejanske kršitve (označujemo kot tip kršitve). Takšna anotirana funkcija vključuje parameter, ki vsebuje opis določene kršitve. Ta opis označuje kontekst kršitve (angl. Violation Context), ki hrani informacije

o vseh strankah vključenih pri kršitvi, tip kršitve, metriko QoS ter SLO, ki je zaznal kršitev, zagotovila akcij (angl. Action Guarantees), kršitve in obligacije za vključene stranke. Funkcija povratnega klica je lahko definirana znotraj komponent poslovnega sloja (npr. komponente EJB) ali spletnega sloja (npr. Servleti, komponente JSP in JSF). V primeru, da te komponente ne operirajo znotraj dogovorjenih nivojev storitve in posledično povzročijo prekoračitev praga opozorila ali dejanske kršitve, morajo biti izvedene določene akcije specificirane znotraj dogovora. Če komponenta aplikacije potrebuje notifikacijo, na primer ko instanca storitve postane nerazpoložljiva, mora komponenta registrirati funkcijo povratnega klica. V primeru prekoračitve praga kršitve, se znotraj aplikacije pokliče funkcija povratnega klica. Ko registriramo funkcijo povratnega klica, lahko aplikacija specificira podatke, ki jih ob klicu funkcije povratnega klica podamo kot parametre (tj. kontekst kršitve). Očitno je, da programske zmožnosti za odzivanje na detektirane kršitve SLA, predstavljajo pomembno vrednost za razvijalce programske opreme, saj omogočajo reagiranje na prekoračitev opozorilnih pragov in pragov kršitev.

4.3.2 Parametri elastičnosti

Med parametre elastičnosti spadajo naslednji parametri: (1) prameter za ročno elastičnost, ki specificira začetno, minimalno ter maksimalno število slojev poslovne logike, vrstni red postavitve (angl. Deployment Order), zakasnitev začetnega nalaganja (angl. Boot Delay) ter zakasnitev zaustavitve (angl. Stop Delay), (2) analogen parameter za spletni sloj ter (3) parameter za definiranje uporabniških pravil za elastičnost (angl. User-Defined Elasticity Rules), ki temeljijo na metrikah, relevantnih za celotno aplikacijo (aplikacija prikazana na sliki 4.6). Vsi trije parametri predstavljajo temelj za preslikavo v platformsko-specifične programske direktive in politike. Potrebno je poudariti, da se doktorska disertacija osredotoča na elastičnost na nivoju računske moči (angl. Compute Elasticity), medtem ko sta elastičnost shrambe in podatkovne baze izven disertacije.

Najbolj opazna pomanjkljivost mnogih sistemov v oblaku je pomanjkanje celostne podpore za elastičnost [68]. Četudi lahko največje obremenitve (angl. Peak Loads) pravilno napovemo, brez podpore elastičnosti, organizacije IT zapravljajo vire med časom majhnih obremenitev (angl. Nonpeak Times). Omenjeni problemi kažejo na pomembnost elastičnosti (tako samodejne kot ročne) v okolju PaaS. Ta logična skupina parametrov naslavlja aspekte elastičnosti pri aplikacijah, specifičnih za okolje oblaka,



Slika 4.6

Topologija storitve oblaka – večslojne aplikacije.

pri čemer politike elastičnosti odrejajo, kako in kdaj so viri dodani ali odstranjeni iz okolja oblaka.

Skaliranje poslovnega/spletnega sloja

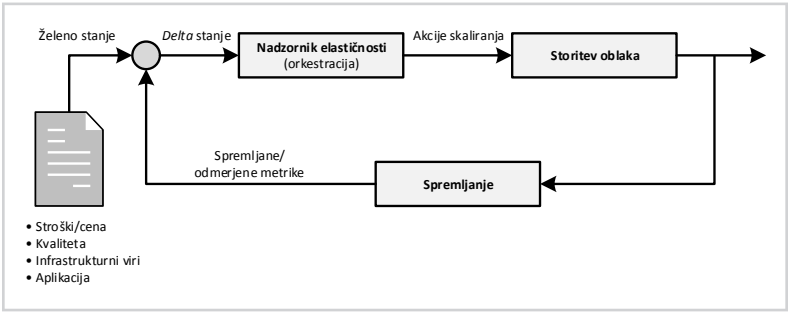
Eden izmed načinov specificiranja politik elastičnosti je definicija začetnega, minimalnega in maksimalnega števila instanc, kar omogoča ročno skaliranje virtualnih strojev v storitvi oblaka, tako v času postavitve, kot v času izvajanja (konfiguracija je prikazana na sliki 4.6). V ta namen vidik ročne elastičnosti vpeljemo na nivo aplikacije in platforme preko predlaganih parametrov (tj. politike). Začetno število definira privzeto vrednost, ki vključuje število instanc VM-jev za določene sloje storitve, ko je aplikacija postavljena, tj. čas postavitve (angl. Deployment Time). Ko je storitev enkrat postavljena, začetna vrednost vnovičnih definicij omogoča modifikacije števila instanc sloja storitve v času izvajanja (angl. Runtime). Pri čemer minimalno in maksimalno število specificirata zgornjo in spodnjo mejo instanc za določen sloj storitve (tj. poslovni ali spletni sloj). Takšne omejitve pravzaprav pridejo prav v privatnih namestitvah oblaka, kjer bi želeli omejiti obseg instanciranja (angl. Instantiation Range) za določenega uporabnika, zaradi omejene narave infrastrukturnih virov. Ta mehanizem definira omejitve elastičnosti za dinamično razširitev in krčenje gruče in omogoča dinamično skaliranje instanc virtualnih strojev v času postavitve in v času izvajanja. Poleg tega vpeljujemo vrstni red postavitve, zakasnitev začetnega nalaganja ter zakasnitev zaustavitve. Vrstni red postavitve definira vrstni red postavitve določenega sloja storitve. Primer: sloj po-

datkovne baze mora biti postavljen najprej, saj ostali sloji potrebujejo povezljivost do baze preko IP naslova. Zakasnitev začetnega nalaganja in zakasnitev zaustavitve specificirata zakasnitveni časovni interval za začetno nalaganje ter zaustavitev določenega sloja storitve oblaka, da bi se izognili konfliktom odvisnosti med temi sloji.

Obstoječi modeli izvajalnih okolij od razvijalcev aplikacij zahtevajo, da so pri postavitvi aplikacije vsi zahtevani viri samo-oskrbovani (angl. Self-provisioned) in inicializirani. V ta namen predlagamo politike za specificiranje virov (npr. persistenca, izenačevanje obremenitev, sporočilne vrste, sporočanje, predpomnjenje itn.), ki morajo biti oskrbljeni. Konfiguracija virov je opredeljena s pomočjo politik. Na primer v primeru oskrbovanja in konfiguracije persistence, lahko determinirano tip podatkovnega vira (angl. Data Source). Sem spadata na primer relacijska podatkovna baza ali baza NoSQL ter njuni parametri (npr. število sočasnih povezav, ime vira podatkovne baze itn.). Podatkovni vir ima običajno nabor lastnosti, ki vključuje informacije, kot so lokacija strežnika podatkovne baze, ime podatkovne baze in ime protokola za komunikacijo s strežnikom (platforma Java EE uporablja termin DataSource [177], medtem ko Mircosoftova platforma .NET uporablja tehnologijo ADO.NET [178]). V sekciji izenačevanja obremenitev lahko na primer specificiramo algoritme za izenačevanje obremenitev (angl. Load Balancing Algorithms), ki ponujajo strežniško afiniteto (angl. Server Affinity) za aplikacijske komponente. Cilj strežniške afinitete je minimizirati število odprtih vtičnic IP (angl. IP Sockets) med zunanjimi aplikacijskimi odjemalci in strežniškimi instancami v gruči. Standardni algoritmi za izenačevanje obremenitev (npr. Round-Robin, Weight-Based, Random Load Balancing [179]) so lahko specificirani znotraj te sekcije, ki na tak način upoštevajo obstoječe povezave med zunanjimi odjemalci in instancami strežnika pri razporejanju bremena odjemalca med instancani slojev storitve oblaka. Skratka sekcija zahtev po virih omogoča samodejno oskrbovanje podatkovnih virov, sporočanja in drugih storitev, ki so zahtevane za izpolnitev pogodbe za gostitev in postavitev aplikacij, specifičnih za oblak.

Parameter za samodejno elastičnost

Drug način specificiranja politik za elastičnost predstavlja uporabo nabora pravil v obliki politik (angl. Policy Rules), ki sledijo paradigmi ECA (Event Condition Action). Takšen pristop označujemo kot samodejno elastičnost. Fundamentalen konstrukt te paradigme so reaktivna pravila v obliki *Ob* Dogodku *Če* Pogoj *Izvedi* Akcijo. Kar pomeni, da ko se zgodi Dogodek in če je Pogoj izpolnjen, se izvede določena Akcija. V



Slika 4.7

Konvencionalna teorija nadzornega sistema aplicirana na elastičnost aplikacij v oblaku.

našem delu adaptiramo paradigmo ECA za vpeljavo samodejne elastičnosti na aplikacijskem in platformskem nivoju, z uporabo politik, ki temeljijo na pravilih (angl. Rule-Based Policies). V tem primeru so pravila sestavljena iz niza pogojev in če so izpolnjeni, le-ti sprožijo določene akcije, specifične za oblak. Vsak pogoj upošteva dogodek ali metriko spremljanja, ki je primerjana (z uporabo modifikatorja) s specifičnim pragom. Informacije o vrednostih teh metrik in dogodkov so ponujene s strani sistema za spremljanje, ki nadzira infrastrukturne vire, aplikacijo, kakovost in stroške.

Predlagana rešitev se poslužuje nadzornega sistema z zaprto zanko (angl. Closed Loop Control Systems) [180]. Le-ta ponuja povratno informacijo aktualnega stanja sistema in jo primerja z želenim stanjem sistema ter skladno s tem izvaja prilagoditve sistema. Podrobneje sistem za spremljanje neprestano spremlja uporabniško izkušnjo ter druge relevantne metrike. V primeru da se odmerjene metrike povzpnejo nad določen prag, ki ga imenujemo fiksna točka (angl. Set Point)(npr. odzivni čas > n sekund), nadzornik elastičnosti (angl. Elasticity Controller) ustrezno skalira okolje in s tem zadosti potrebnim pogojem. Nadalje je odmerjena metrika (npr. odzivni čas) primerjana z novimi viri in v primeru da je še zmeraj nad določenim stanjem, nadzornik skalabilnosti nadaljuje s skaliranjem storitve oblaka. Ko se odmerjena metrika spusti pod želenih n sekund, nadzornik elastičnosti skalira okolje tako, da zmanjša število uporabljenih virov in nadaljuje s preverjanjem ali je odmerjena metrika znotraj sprejemljivega območja. Da bi se sistem izognil konstantnim preiskovanjem fiksne točke, morajo biti informacije spremljanja skupaj s fiksno točko nadzorovane. Slednje omogoča infrastrukturi, da operira bolj učinkovito in na bolj stabilen način. To je realizirano s postavitvijo pasu (angl. Band) okoli fiksne točke tako, da definiramo del-

ta stanje in izvajamo akcije zgolj takrat, ko odmerjena zmogljivost sistema pade izven določenega pasu. Takšen proces elastičnosti na nivoju aplikacije je prikazan na sliki 4.7.

Skaliranje storitve oblaka je izvedeno s strani storitvenega orkestratorja (tj. nadzornik elastičnosti). Predpostavimo, da je podatkovna baza neskončno skalabilna za podporo spletnega in poslovnega sloja, saj je skalabilnost na nivoju podatkovnega sloja kompleksen proces in iz tega razloga izven obsega te disertacije. Naš pristop upošteva trenutno stanje vseh aplikacijskih slojev storitve oblaka, da bi identificirali, kateri sloji so odgovorni za degradacijo (angl. Underprovisioned) oz. kateri del storitve oblaka je trenutno preveč oskrbljen (angl. Overprovisioned). Kot rezultat tega nadzornik skalabilnosti:

1. identificira potencialna ozka grla,
2. determinira ali je ozka grla povzročila težava s performanco, kapaciteto ali stroški/ceno,
3. lokalizira ozka grla znotraj storitve oblaka,
4. sproži ustrezno operacijo skaliranja (npr. povečaj število instanc poslovnega sloja – operacija scaleUp).

Z upoštevanjem teh dejstev, lahko predlagan pristop reducira stroške, ki so nastali s strani uporabnika infrastrukturnih storitev oblaka, medtem ko upošteva uporabniško izkušnjo in omogoča skalabilnost storitve zgolj na nivojih z ozkimi grli. Naš predlog torej vpeljuje kombiniran pristop samodejnega skaliranja, ki upošteva stroške, kakovost in delovne obremenitve (angl. Combined Cost-Aware, Quality-Aware and Workload-Adaptive Approach) in kot tak predstavlja ključnega diferenciatorja pred večino sistemov za doseganje samodejne elastičnosti v povezavi z obstoječimi pristopi.

4.4 *Evalvacija: preslikava parametrov v sistem metapodatkov Java EE*

Za splošno-namenske programske jezike, kot so Java, C# in Python, lahko implementiramo parametre z uporabo jezikovno-specifičnih politik in anotacij, ki jih v nekaterih znanstvenih delih imenujejo kar programske direktive (angl. Programming Directives). Primeri takšnih direktiv so Java Annotation [45], C# Attribute Declaration [181]

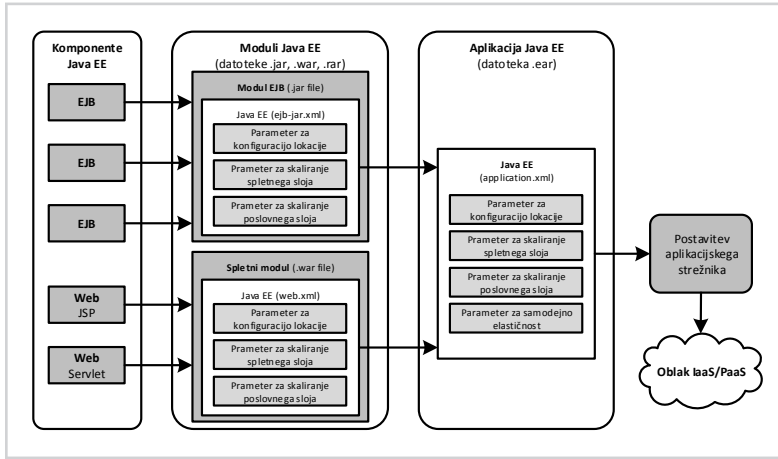
in Python Decorators [182]. Politike skupaj s programskimi direktivami v našem delu označujemo kot parametre, ki jih lahko preslikamo v poljuben platformsko-specifičen programski jezik in vmesno programsko opremo (angl. Middleware). Ker bo Javanska platforma, t. i. platforma izbire pri številnih oblakih PaaS [46], se naše delo osredotoča na Java EE Annotation Metadata, ki jih izpeljemo iz platformsko neodvisnih parametrov. Preslikava med sistemom parametrov (angl. Parameter System) in sistemom metapodatkov (angl. Metadata System) je realizirana na štiri načine:

1. deklaracija novih anotacijskih metapodatkov,
2. razširitev globalnega postavitvenega deskriptorja (*application.xml*),
3. razširitev postavitvenega deskriptorja *ejb-jar.xml*,
4. razširitev postavitvenega deskriptorja *web.xml*.

Aplikacija Java EE je logična kolekcija enega ali več modulov Java EE, ki jih med seboj povezujejo postavitveni deskriptorji aplikacije [183]. Postavitveni deskriptor, dokument XML s končnico .xml, opisuje konfiguracije postavitve aplikacije, modula ali komponente. Komponente so lahko sestavljene (angl. Assembled) in postavljene na nivoju modula ali nivoju aplikacije. Modul Java EE je sestavljen iz ene ali več komponent Java EE za isti tip vsebnika in opcijsko, enega postavitvenega deskriptorja komponente za posamezen tip vsebnika [43]. Moduli Java EE so lahko različnih tipov:

- poslovni modul (angl. Enterprise Module),
- moduli EJB (angl. EJB Modules),
- spletni moduli (angl. Web Modules),
- moduli aplikacijskega odjemalca (angl. Application Client Modules),
- moduli adapterja virov (angl. Resource Adapter Modules).

Ker se v doktorski disertaciji osredotočamo zgolj na spletni sloj ter sloj poslovne logike, so moduli aplikacijskega odjemalca in moduli adapterja virov izven obsega in jih ne bomo obravnavali. Poslovni modul lahko vsebuje nič ali več spletnih modulov, nič ali več modulov EJB ter druge skupne ali zunanje knjižnice. Zato da se postavitev



Slika 4.8

Združitev in postavitev storitve oblaka z upoštevanjem parametrov.

teh raznolikih modulov zgodi simultano in koherentno [157], je vse to zapakirano v arhiv EAR (Enterprise Archive). Opcijski postavitveni deskriptor poslovnega modula je definiran znotraj datoteke *application.xml*, ki jo označujemo kot globalni aplikacijski deskriptor (angl. Global Deployment Descriptor). Aplikacijski deskriptor razširimo za podporo (1) parametru za konfiguracijo lokacije, (2) parametru za skaliranje poslovnega sloja, (3) analognemu parametru za skaliranje spletnega sloja ter (4) parametru za samodejno elastičnost.

Ker sta spletni modul ter modul EJB lahko postavljena ločeno, izven aplikacije same, ponujamo razširitve postavitvenih deskriptorjev *ejb-jar.xml* in *web.xml* na analogen način, z izjemo parametra za samodejne elastičnosti. Parameter za samodejne elastičnosti izključujemo iz spletnega modula in modula EJB, saj mora upoštevati aplikacijo kot celoto in kot tak ni primeren za aplikabilnost nad enim samim modulom. Upoštevajte, da morajo biti razširitve spletnega modula in modula EJB uporabljene izključno za postavitev izven poslovnega modula, saj se na tak način izognemo potencialnim konfliktom postavitve in odvisnosti. Zaradi tega so komponente EJB sestavljene v datoteko Java Archive (JAR) s postavitvenim deskriptorjem *ejb-jar.xml*. Spletne komponente so sestavljene v datoteko Web Archive (WAR) s postavitvenim deskriptorjem *web.xml*. Proces sestavljanja in postavitve storitve oblaka, ki ustreza predlaganim parametrom, je prikazan na sliki 4.8.

```
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface Monitor {
  MonitorCategory monitorCategory() default monitorCategory.NONE;
  MetricType metricType() default MetricType.NONE;
  int timePeriod() default 30;
};
```

Slika 4.9

Deklaracija anotacijskega tipa Monitor.

```
@Monitor(monitorCategory=INFRASTRUCTURE.COMPUTE, metricType=CPU_UTILIZATION, timePeriod=60)
private void cpuUtilizationMetricUpdated(MonitorContext context) {
  ...
}
```

Slika 4.10

Primer uporabe anotacijskega tipa Monitor.

4.4.1 Metapodatki za nadzor

Eden izmed ciljev doktorske disertacije je oblikovati rešitev za nadzor aplikacij Java EE, ki so specifične za oblak in upoštevajo parametre za nadzor, opisane v prejšnjem poglavju. V ta namen predlagamo nabor metapodatkov za nadzor, ki jih izpeljemo iz parametrov za nadzor. Ta skupina metapodatkov vključuje deklaracijo anotacije za spremljanje aplikacije, razširjeno shemo (`http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/application_7.xsd`) postavitvenega deskriptorja za parameter lokacijske konfiguracije, ter deklaracijo anotacije za kršitve SLA.

Metapodatki za spremljanje aplikacije

Dodatno predlagamo nov anotacijski tip, ki je zadolžen za pridobivanje različnih infrastrukturnih, aplikacijskih in stroškovnih metrik ter metrik kakovosti. Izvorna koda 4.9 prikazuje deklaracijo predlaganega anotacijskega tipa Monitor. Atribut *monitorCategory* specificira tip/kategorijo specifične metrike, medtem ko *timePeriod* definira časovno periodo merjenja povprečne vrednosti metrike. Primer uporabe anotacijskega tipa Monitor prikazuje izvorna koda 4.10. V tem primeru je metrika povprečne izkoriščenosti CPU-ja injicirana v objektni tip *MonitorContext* in je na razpolago za nadaljnjo uporabo v telesu metode *cpuUtilizationMetricUpdated*.

Metapodatki za lokacijske konfiguracije

Določeni tipi aplikacij zahtevajo zmožnost določanja, ali sta lahko dva ali več slojev postavljena tesno skupaj, saj se zanašata na hitro komunikacijo ali skupne odvisnosti strojne opreme, kot je na primer zanesljiva komunikacijska povezava (angl. Communication Link). Drugi tipi aplikacije zahtevajo zmožnost konfiguriranja geografsko ločenih postavitev dveh ali več slojev, zaradi zagotavljanja visoke stopnje razpoložljivosti ali okrevanja po katastrofi (angl. Disaster Recovery) [184].

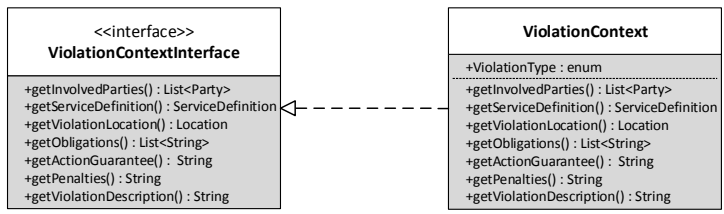
Skladno s tem razširimo shemo postavitvenega deskriptorja Java EE z dodatnimi lokacijskimi konfiguracijami. Primer uporabe v izvorni kodi 3 prikazuje postavitveni deskriptor *application.xml* (skladen s shemo XML), ki je razširjen za podporo lokacijskih konfiguracij. Aplikacija Java EE je ločena v dva modula: spletni modul in modul EJB. Moduli so definirani v postavitvenem deskriptorju v obliki elementa <*module*>, ki predstavlja element kompleksnega tipa (angl. Complex Type Element). Modul *moduleType* definira enojni modul Java EE in vsebuje elemente *connector*, *ejb*, *java*, ali *web*. Modul *moduleType* razširimo s podelementom <*cloud-location*>. Le-ta specificira priporočljivo lokacijo podatkovnega centra za izvajanje posameznega sloja aplikacije. Primer: izvorna koda 4.11 prikazuje konfiguracijo strategije postavitve in konfiguracijo prednostne lokacije za izvajanje EJB in spletnih modulov (oba modula bosta v tem primeru postavljena v isto podregijo EU-Ireland). Element <*deployment-strategy*> vključuje atribut politike z vrednostjo *best-performance* ter en podelement zadolžen za opis ustrezne strategije postavitve. Konfiguracija poskuša obdržati virtualne stroje (spletni sloj in sloj EJB) čim bližje drug drugemu, na primer na istem fizičnem gostitelju ali znotraj istega območja razpoložljivosti, ki zagotavlja hitro omrežno povezljivost med posameznimi vozlišči.

V nadaljevanju definiramo sekcijo <*cloud-location*>, ki vsebuje elementa <*region-name*> ter <*preferred-subregion*>. Slednji vsebuje enega ali več elementov <*subregion*>, ki specificira priporočljivo lokacijo podatkovnega centra za izvajanje posameznega dela storitve oblaka. Element <*subregion*> ravno tako vključuje elementa <*availability-zone*> ter <*affinity-group*>. Element <*availability-zone*> definira določeno območje znotraj podatkovnega centra. Uporaba elementa <*affinity-group*> omogoča povezovanje virov znotraj istega podatkovnega centra z namenom zagotavljanja boljše zmogljivost ali skladnosti s predpisi (angl. Regulatory Compliance). V našem primeru element <*affinity-group*> povezuje vire iz spletnega modula in modula EJB in jih postavlja v isto

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns=http://xmlns.jcp.org/xml/ns/javaee xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/application_7.xsd
version="7">
  <display-name>Application.ear</display-name>
  <description>Application description.</description>
  ...
  <deployment-strategy policy="best-performance">
    <description>Deployment strategy for a group of VMs</description>
  </deployment-strategy>
  ...
  <module id="WebModule_1">
    <web>
      <web-uri>WebModule.war</web-uri>
      <context-root>/</context-root>
      ...
    </web>
    ...
    <cloud-location>
      <region-name>EU</region-name>
      <preferred-subregion>
        <subregion>
          <name>EU-Ireland</name>
          <availability-zone>EU-Ireland-1A</availability-zone>
          <affinity-group>GROUP-1A</affinity-group>
        </subregion>
      </preferred-subregion>
    </cloud-location>
    ...
  </module>
  <module id="EjbModule_1">
    <ejb>EjbModule.jar</ejb>
    ...
    <cloud-location>
      <region-name>EU</region-name>
      <preferred-subregion>
        <subregion>
          <name>EU-Ireland</name>
          <availability-zone>EU-Ireland-2A</availability-zone>
          <affinity-group>GROUP-1A</affinity-group>
        </subregion>
      </preferred-subregion>
    </cloud-location>
    ...
  </module>
  ...
</application>
```

Slika 4.11

Razširitev postavitvenega deskriptorja application.xml s parametrom za lokacijske konfiguracije.



Slika 4.12

UML razredni diagram ViolationContext.

območje razpoložljivosti (to sta lahko območji EU-Ireland-2A ali EU-Ireland-1A), tako da specificira isto vrednost elementa <*affinity-group*> (tj. GROUP-1A) in definira primerne postavitvene strategije. Na podoben način sta lahko razširjena tudi postavitvena deskriptorja *ejb-jar.xml* in *web.xml* s strategijami postavitve in prednostnimi izvajalnimi lokacijami za vsebnike EJB in spletne vsebnike. Da bi se izognili morebitnim postavitvenim konfliktom in konfliktom odvisnosti, mora biti razširitev modula EJB in spletnega modula uporabljena zgolj za postavitev izven poslovnega modula. Takšna naredba velja za vse ekvivalentne scenarije v prihajajočih poglavjih.

Metapodatki za kršitve SLA

Predlagamo deklaracijo anotacijskega tipa @PostSLAViolation, ki ga uporabimo kot mehanizem povratnega klica za signaliziranje, da je prišlo do kršitve dogovora SLA (prekoračitev opozorilnega praga ali praga kršitve). V ta namen je anotirana metoda zadolžena za detektiranje in ustrezno odzivanje na določene kršitve. Deklaracija anotacije ne zahteva atributov. Metoda povratnega klica je lahko definirana zgolj znotraj komponent EJB in spletnih komponent (npr. JSP-ji, JSF-ji, Servleti).

Izvorna koda 4.13 prikazuje deklaracijo opisanega anotacijskega tipa, medtem ko izvorna koda 4.14 prikazuje primer njene uporabe znotraj sejnega zrna *CustomerManagerSB*. V tem primeru vsebuje anotirana metoda parameter tipa *ViolationContext*, ki hrani podatkovni opis določene kršitve SLA, vključno z vsemi udeleženci kršitve, definicijo storitve, tipom kršitve, lokacijo kršitve, zagotovila akcij (angl. Action Guarantees), kazni in obveznosti vpletenih strank. Struktura razreda *ViolationContext* je prikazana na sliki 4.12. Telo metode *notify(ViolationContext):void* lahko vsebuje poljubno implementacijo, od proženja notifikacij preko elektronske pošte, klicanja

```
@Documented
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface PostSLAViolation {
};
```

Slika 4.13

Deklaracija anotacijskega tipa PostSLAViolation.

```
@Stateless
public class CustomerManagerSB {

@PostSLAViolation
void notify(ViolationContext context) {
    System.out.println("An SLA violation has occurred.");
    System.out.println(context.getViolationDescription());
    ...
    }
}
```

Slika 4.14

Primer uporabe anotacijskega tipa PostSLAViolation znotraj sejnega zrna EJB.

ustrezne operacije za skaliranje (npr. operaciji scaleUp in scaleDown), vse do klicanja poljubnega orkestracijskega delovnega toka v obliki spletnih storitev REST/SOAP. Predpostavimo, da sejno zrno EJB potrebuje notifikacijo (npr. instanca sloja storitve preneha delovati). V tem primeru mora komponenta registrirati metodo povratnega klica *notify(ViolationContext):void* in v primeru, da pride do kršitve SLA-ja, je ta metoda prožena znotraj aplikacije. Omenjen pristop omogoča lokalizacijo kršitev SLA-jev znotraj storitve oblaka. Primer: ozko grlo (angl. Bottleneck) se je pojavilo v eni izmed instanc poslovnega sloja zaradi degradacije performance.

4.4.2 Metapodatki elastičnosti

Skupina metapodatkov elastičnosti vključuje metapodatke, ki specificirajo začetno, minimalno ter maksimalno število slojev poslovne logike (tj. vsebnikov EJB), analogne metapodatke za določanje števila spletnih slojev (tj. vsebnikov EJB), ter metapodatke za definiranje uporabniških pravil za elastičnost. Vsi trije metapodatki so izpeljani iz parametrov za elastičnost ter so preslikani v globalni postavitveni deskriptor v obliki razširitve sheme XML. Elastičnost v kontekstu računalniških oblakov mora biti naslovljena iz dveh perspektiv: perspektive ponudnika in perspektive končnega uporabni-

ka. Iz perspektive ponudnika mora elastičnost zagotoviti boljšo uporabo računalniških virov, varčevanje z energijo, ponujanje ekonomije obsega in omogočanje simultane strežbe več uporabnikom, medtem ko mora biti iz perspektive uporabnika, elastičnost uporabljena za preprečitev nezadostnega oskrbovanja virov, degradacije zmogljivosti sistema in povišanja stroškov [84].

Skaliranje EJB/spletnega sloja

Predlagamo razširitev sheme z definicijo začetnega, minimalnega in maksimalnega števila instanc poslovnega sloja (tj. vsebnikov EJB), ki jih je potrebno oskrbeti, ko je storitev oblaka postavljena v oblak. Naša predlagana rešitev omogoča vnovično definicijo (angl. Redefinition) politik ročne elastičnosti v času izvajanja (npr. vnovična definicija začetnega števila instanc vsebnikov EJB), kar omogoča adaptacijo obnašanja storitve oblaka, ko je ta že enkrat postavljena. Ker so informacije v postavitvenih deskriptorjih deklarativne, so lahko spremenjene brez potrebnih modifikacij oz. poseganja v izvorno kodo. V času izvajanja, strežnik Java EE prebere te informacije in ustrezno reagira [183]. Z upoštevanjem teh dejstev definiramo nov element <*scaling-policy*>, ki vključuje enega ali več elementov <*tier*>. Element <*tier*> vsebuje atribut *type*, ki je lahko tipa *EJB*, *Web* ali *Database*. Vsak element <*tier*> definira štiri podelemente: <*deployment-order*>, <*boot-delay*>, <*stop-delay*> ter <*instances*>. Element <*deployment-order*> specificira vrstni red postavitve določenega sloja storitve (npr. naslov IP podatkovne baze je znan šele v času njene postavitve). Iz tega razloga mora biti sloj podatkovne baze postavljen najprej, saj sloj EJB potrebuje povezljivost do baze preko naslova IP. Spletni sloj lahko nato postavimo na koncu, saj zahteva povezljivost in dostop do sloja EJB. Elementa <*boot-delay*> ter <*stop-delay*> specificirata zakasnitveni časovni interval (v sekundah) za začetno nalaganje ter zaustavitev določenega sloja storitve oblaka, da bi se izognili konfliktom odvisnosti med temi sloji. Element <*instances*> definira privzeto, minimalno ter maksimalno število instanc za posamezni sloj. Vrednosti atributov *minimum* in *maximum* morajo biti nenegativna števila, pri čemer mora biti vrednost atributa *minimum* manjša ali enaka vrednosti atributa *maximum*. Vrednost atributa *minimum* je lahko enaka nič, v primeru da storitev oblaka ne vsebuje instanc določenega sloja. Element *minimum* je opcijski in se v primeru njegove odsotnosti predpostavlja, da ima vrednost enako ena. Podobno je če atribut *maximum* ni prisoten, pri čemer se predpostavlja, da ima neomejeno vrednost (angl. Unbounded). Atribut *initial* je zahtevan in mora vsebovati vrednost znotraj obsega definiranega s

strani atributov *minimum* in *maximum*. Takšna politika ročne elastičnosti mora biti specificirana znotraj poslovnega modula (tj. arhiv EAR) v postavitvenem deskriptorju *application.xml*, ali znotraj modula EJB v postavitvenem deskriptorju *ejb-jar.xml*. Upoštevajte, da v slednjem primeru element <*tier*> ni prisoten, saj so politike za skalabilnost uveljavljene znotraj EJB-specifičnega postavitvenega deskriptorja. Kot take vpeljujejo aspekte ročne elastičnosti nad vsemi komponentami EJB, ki so vključene v sekciji <*enterprise-beans*>. Komponente EJB predstavljajo sejna zrna (angl. Session Beans), entitetna zrna (angl. Entity Beans) ali sporočilna zrna (angl. Message-Driven Beans). Nekateri komercialni aplikacijski strežniki (npr. WebSphere Application Server [44], JBoss AS [185] itn.) vpeljujejo lastniške deskriptorje, ki omogočajo definiranje minimalnega in maksimalnega števila instanc zrn vzdrževanih v bazenu instanc. Na drugi strani naš predlog naslavlja omejevanje instanc, ki temeljijo na vsebnikih (angl. Container-Level Instances). V tem primeru je skalabilnost implementirana z instanciranjem/sproščanjem nedeljenih vsebniških instanc (preko replikacije), kjer se lahko izvajajo številne replike komponent razvijalcev, tj. horizontalna skalabilnost, ki temelji na vsebnikih [48]. Podobna analogna razširitev je predlagana za namen spletnega vsebnika (angl. Web Container), ki omogoča definicijo začetnega, minimalnega in maksimalnega števila instanc spletnih vsebnikov (tj. instance spletnega sloja), vrstnega reda postavitve, zakasnitve začetnega nalaganja ter zakasnitve zaustavitve. Posledično definiramo sekcijo <*scaling-policy*>, ki specificira spletni vsebniški tip, vključno z vsemi prej omenjenimi aspekti. Takšna uvedba je lahko specificirana znotraj projekta EAR v postavitvenem deskriptorju *application.xml*, ali znotraj spletnega modula v postavitvenem deskriptorju *web.xml*. V drugem primeru je element <*tier*> prisoten zaradi istih razlogov, ki so izpostavljeni v prej omenjeni razširitvi *ejb-jar.xml*. V nadaljevanju definiramo element <*self-provisioning*>, ki omogoča opis vseh virov, zahtevanih s strani aplikacije, še preden je le-ta postavljena v oblak (npr. število sočasnih povezav do podatkovne baze, ime vira JDBC, tovarna povezav JMS, viri JNDI, predpomnjenje, izenačevanje obremenitev itn.). Ta razširitev je lahko definirana znotraj globalnega postavitvenega deskriptorja, postavitvenega deskriptorja EJB in spletnega postavitvenega deskriptorja, kar omogoča samodejno oskrbovanje zahtevanih aplikacijskih virov za določen sloj aplikacije kot celote.

Metapodatki za samodejno elastičnost

Samodejno skaliranje omogoča konzumiranje infrastrukturnih virov, ko so le-ti potrebni, in vračanje teh virov v bazen, ko ti viri niso več potrebni. Takšna oblika končnim uporabnikom ponuja samodejno odzivnost na incidente zmogljivosti/kapacitete in odzivnost na zahteve, ki so povezane z viri, stroški in kakovostjo. S tem se zmanjšajo stroški dostave storitve in izpadi zaradi človeških napak [84]. V našem primeru je samodejna elastičnost realizirana s funkcionalnostjo horizontalne skalabilnosti oblakov PaaS. Predlagane razširitve sheme postavitvenega deskriptorja indicirajo uporabniško definirana pravila za elastičnost. Ta pravila nam omogočajo regulacijo obnašanja aplikacije, in sicer ali naj le-ta samodejno skalira ter pod katerimi pogoji. Pri tem upošteva kombinirane vire (iz nivoja infrastrukture in aplikacije), stroške ter kakovost. Nadzornik elastičnosti kontinuirano preverja stanje storitve in ga primerja z uporabniško definiranimi pravili. Ko so enkrat pogoji proženja izpolnjeni, nadzornik aktivira posledične akcije. Pravila so definirana znotraj elementa <*autoscaling-policy*>, ki vsebuje številne podelemente <*condition*> ter enega ali več elementov <*action*>, ki skupaj zadostujejo paradigmi ECA. Primer razširitve postavitvenega deskriptorja je prikazan v izvorni kodi 4.15. Da bi naslovili aspekt stroškov in kakovosti, definiramo ločen element <*business-constraints*>, ki vsebuje enega ali več elementov <*constraint*>. Ta element nosi atribut *id*, ki je lahko tipa *cost* ali tipa *end-user-experience*. Poslovne omejitve delujejo v simbiozi s pogoji na nivoju metrik virov. Na primer ko je pogojem zadoščeno, se morajo le-ti podrediti poslovnim omejitvam (z upoštevanjem stroškov in kakovosti), preden sprožijo ustrezne akcije. Ker je računalništvo v oblaku dostavljeno kot storitev (npr. elektrika in voda), se od lastnikov aplikacij pričakuje, da bodo porabili minimalne stroške za doseganje določene performance aplikacije [137]. V ta namen trdimo, da je stroškovni kriterij (angl. Cost-Aware Criterion) eden izmed pomembnejših elementov elastičnosti aplikacij za oblak. Da bi preprečili sistem pred konstantnim iskanjem specifičnega praga, naš predlog inkorporira pas okoli fiksne točke tako, da definira delta stanje in izvede akcije zgolj takrat, ko odmerjena performanca pade izven določenega pasu. Cilj te razširitve je omogočiti aplikaciji oz. storitvi oblaka, da ohrani avtonomnost v okviru skladnosti želenega nivoja storitve in na tak način doseže zahteve aplikacije po kakovosti, stroških in zmogljivosti. Če je le-to potrebno, so dodani/odstranjeni dodatni viri, da bi ohranili skladnost in ustregli povišanim/znižanim zahtevam.

Predlagane razširitve za skaliranje, ki temeljijo na politikah, alocirajo dodatne stre-

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns=http://xmlns.jcp.org/xml/ns/javaee xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/application_7.xsd
version="7">
   <display-name>Application.ear</display-name>
   <description>App description</description>
   ...
   <autoscaling-policy>
      <business-constraints>
         <constraint id="cost">
            <available-budget>50</available-budget>
            <currency>EUR</currency>
            <time-period>MONTH</time-period>
         </constraint>
         <constraint id="end-user-experience">
            <required-response-time>5</required-response-time>
         </constraint>
         ...
      </business-constraints>
      <rule id="r-01">
         <condition id="resource">
            <metric state="average">POOL_SIZE</metric>
            <modifier>greaterOrEqual</modifier>
            <threshold>95</threshold>
            <timePeriod>60</timePeriod>
         </condition>
         ...
         <action>
            <trigger-name>deployBusinessTierInstance</trigger-name>
            <instances count="1" />
         </action>
         ...
      </rule>
      ...
   </autoscaling-policy>
   ...
</application>
```

Slika 4.15

Razširitev postavitvenega deskriptorja application.xml s samodejno (reaktivno) elastičnostjo.

žnike pri skaliranju kadarkoli določene performančne metrike presežejo prag, in umaknejo redundantne strežnike pri skaliranju navzdol, kadarkoli so te metrike manjše od praga. Predvidevamo, da imajo aplikacijski administratorji zadostno znanje o izvajajoči se storitvi oblaka, da bi definirali primerne politike.

4.5 Rezultati in diskusija

Rezultati evalvacije pokažejo, da lahko z novo-predlaganimi anotacijskimi metapodatki uspešno nadziramo šestdeset metrik ter upravljamo z dvainšestdeset novimi elementi XML, ki so manifestacija spodaj ležečih infrastrukturnih operacij oblaka. Te metrike vključujejo eno stroškovno metriko (tj. celotni infrastrukturni stroški za izvajanje aplikacije), dve metriki kakovosti (odzivni čas in razpoložljivost aplikacije), enaintrideset infrastrukturnih metrik (npr. omrežni promet in izkoriščenost CPU) in šestindvajset aplikacijsko-specifičnih metrik (npr. število aktivnih sej in število povezav na bazo). Novi elementi XML, s katerimi upravljamo infrastrukturne operacije in so odraz predlaganega sistema parametrov, vključujejo enajst parametrov za spremljanje aplikacije (npr. Compute, omrežje ter Volume), deset parametrov za konfiguracijo lokacije (npr. strategije postavitve – *najboljša zmogljivost, maksimalna razpoložljivost in najboljši odzivni čas*; ter prednostne podregije), tri parametre za kršitve SLA-jev (opozorilni prag, prag kršitve ter kontekst kršitve), dvaindvajset prametrov ročne elastičnosti (npr. vrstni red postavitve in začetno, minimalno ter maksimalno število instanc) in šestnajst parametrov samodejne elastičnosti, kar vključuje številne parametre v sekciji poslovnih omejitev (npr. definicija razpoložljivih mesečnih stroškov ter zahtevan odzivni čas) ter parametre v sekciji uporabniško-definiranih pravil za skalabilnost (npr. pogoji, metrike, modifikatorji ter akcije). S tem pokažemo, da je administratorjem in razvijalcem aplikacij omogočen večji nadzor in boljše upravljanje aplikacij v oblaku.

"There is only one thing that makes a dream impossible to achieve: the fear of failure."

— Paulo Coelho

5

Upravljanje kompenzacij v orkestracijskih platformah oblaka

5.1 Uvod

Glavni prispevek tega poglavja je vpeljava celovitega upravljanja kompenzacij v orkestracijske platforme oblaka, pri čemer (1) izvedemo arhitekturno analizo obstoječih sistemov za izvajanje delovnih tokov specifičnih za oblak (angl. Cloud-Specific Workflow Systems), (2) definiramo generični pristop za upravljanje kompenzacij, ki so aplikabilne nad orkestratorji oblaka ter (3) predlagamo nov algoritem Compensation Activities Search (CAS) za preiskovanje delovnih tokov orkestratorjev oblaka (tj. usmerjenih grafov) in iskanje primernih aktivnosti, ki jih je potrebno kompenzirati. Da bi dosegli dogodkovno-usmerjeno izboljšavo za upravljanje kompenzacij, predlagamo podporo za upravljavce kompenzacij (angl. Compensation Handlers), ki vključujejo t. i. kompenzacijske dogodke tipa *catch*, kompenzacijske dogodke tipa *throw* in kompenzacijske aktivnosti. Pojem kompenzacijskega dogodka gnezdenih delovnih tokov (angl. Event Subworkflow) je ravno tako podprt. Ker je cilj naše predlagane rešitve vpeljati semantiko upravljanja kompenzacij v orkestracijsko arhitekturo oblaka, adaptiramo izvršljivi BPMN 2.0 (angl. Executable BPMN 2.0) za doseganje tega cilja. Predlagana rešitev je generična in jo lahko adaptirajo ostali izvajalni pogoni delovnih tokov (angl. Workflow Execution Engines) za dosego obogatitev orkestratorjev oblaka s konceptom upravljanja kompenzacij. Da bi pokazali efektivnost in izvedljivost predlaganega pristopa, na koncu prikažemo primer uporabe s tehnologijo BPMN 2.0 za orkestracijo opravil specifičnih za okolje oblaka in razvijemo sistem PoC.

5.2 Upravljanje kompenzacij

Kompenzacija je proces povrnitve oz. ponujanja alternative za uspešno zaključene aktivnosti po tem, ko je prišlo do napake [58]. V svetu inženiringa je kompenzacija planiranje stranskih učinkov ali drugih nepredvidenih težav v zasnovi sistema, kar predstavlja plan "nasprotnega-postopka" za predvidljive stranske učinke, in je izvedena za produciranje bolj učinkovitih in uporabnih rezultatov. Koncept kompenzacije je prisoten na številnih področjih, kot so elektro inženiring (npr. upornik ali induktor se kompenzirata z negativno upornostjo pri razveljavi z odvajanjem plina), gradbeni inženiring (dilatacije na pločnikih, stavbah in mostovih se kompenzirajo s širjenjem in krčenjem) ter v zadnjih časih računalniška/informacijska znanost. V informacijski znanosti je bil pojem kompenzacij prvič omenjen pri Sagah [186] namenjenih za simuliranje transakcijskih lastnosti atomarnosti, konsistence, izolacije in trajnosti za

dolgo-izvajajoče se aplikacije podatkovnih baz, ki bi bili predragi za implementacijo v obliki enojnih transakcij ACID (Atomicity, Consistency, Isolation, Durability). V splošnih procesnih delovnih tokovih so kompenzacije kompleksnejše kot pri Sagah, saj je (1) proces delovnega toka strukturno bolj kompleksen in s tem izvajanje procesa vzpostavi precej zapleten nadzor in odvisnosti med aktivnostmi procesa ter (2) proces delovnega toka običajno vključuje več neodvisnih sistemov podatkovnih baz, aplikacij in uporabnikov [58]. Za vzpostavitev delovnih tokov, tolerantnih na napake, izvajajoči se delovni tokovi zahtevajo uvedbo upravljanja kompenzacij, ki naslavlja proces obnovitve napak. Delovni tokovi so lahko okrnjeni zaradi sistemskih okvar, spremenjenih pogojev delovnih obremenitev, omrežnih napak, napak programske in strojne opreme itn. Pomembna karakteristika storitev oblaka je oskrbovanje nefunkcionalnih zagotovil v obliki SLA-jev [187]. Ker se lahko pri izvajanju številnih scenarijev oblaka pripetijo sistemske napake, se lahko zgodi, da pride do prekršitve vzpostavljenih SLA-jev. Zaradi tega je pomemben napredek pri učinkovitih tehnikah za vzpostavitev delovnih tokov nazaj v konsistentno stanje, da bi se izognili dragim kršitvam SLA-jev (npr. kršitev zagotovil stranke po času izvajanja, stroških ali stopnji razpoložljivosti).

Obstajata dva različna tipa težav ali anomalij, ki se lahko pripetijo med izvajanjem delovnih tokov: izjeme in napake [188]. Izjeme so semantične napake, do katerih lahko pride zaradi sistemskih napak ali zaradi nove nepredvidene situacije, ki jo je povzročilo zunanje okolje, medtem ko so napake sistemske narave in jih lahko povzroči oprema, komunikacijske naprave, ali napake v programih [189]. V primeru nastanka izjem, pogon delovnih tokov ali terminira proces delovnega toka ali pa se vrne v izvajanje procesa delovnega toka, ko je izjema enkrat obravnavana [190]. Medtem upravljavec izjem zajema izjemne dogodke in na njih ustrezno reagira. Obnovitev napak povrne neuspele delovne tokove v semantično sprejemljivo stanje, tako da kompenzira že zaključene aktivnosti, dokler ni doseženo sprejemljivo stanje [58]. Skratka, cilj kompenzacij je povrniti učinke operacij, ki so bile uspešno zaključene, preden je prišlo do napake z razlogom, da bi povrnili delovni tok v konsistentno stanje.

Pojem kompenzacije je že vrsto let prisoten znotraj industrijskih specifikacij, kot sta BPMN [64] in Web Services Business Process Execution Language (WS-BPEL) [65]. Primarni cilj BPMN 2.0, ki je bil leta 2011 razvit s strani konzorcija OMG (Object Management Group), je ponuditi standardno notacijo, ki bi bila razumljiva vsem poslovnim uporabnikom, in sicer:

1. poslovnim analitikom, ki kreirajo začetne osnutke procesov,
2. inženirjem zadolženim za implementacijo tehnologije, ki bo izvedla te procese,
3. domenskim strokovnjakom, ki bodo upravljali in spremljali te procese.

Na drugi strani specifikacija WS-BPEL, ki je bila razvita s strani organizacije OASIS (Organization for the Advancement of Structured Information Standards), definira jezik za izvajanje poslovnih procesov, ki temelji na spletnih storitvah in je namenjen analitikom poslovnih procesov (angl. Business Process Analysts), razvijalcem/arhitektom programske opreme (angl. Software Developers/Architects) ter sistemskim integratorjem (angl. System Integrators). Procesi WS-BPEL in BPMN imajo to prednost, da so izvršljivi in uporabljajo spletne storitve neposredno znotraj procesov.

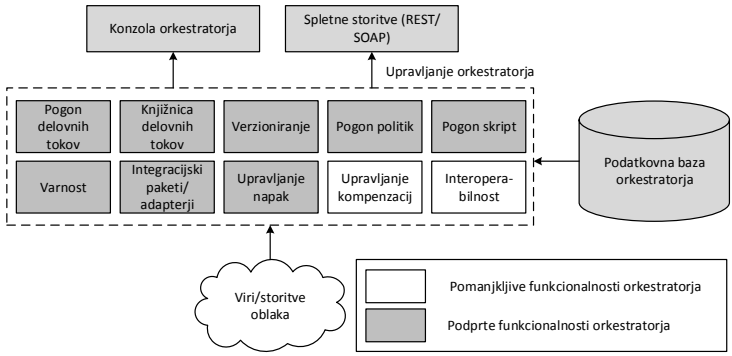
V prvem primeru BPMN 2.0 uporablja upravljavce kompenzacij in kompenzacijske dogodke za doseganje procesnih kompenzacij. Ena izmed razpoznavnih funkcionalnosti BPMN je podpora transakcijam in kompenzacijam v poslovnih procesih [191]. Upravljavec kompenzacij izvaja korake, ki so potrebni za povrnitev efektov aktivnosti, medtem ko imajo v primeru podprocesa, kompenzacijski upravljavci dostop do podatkov podprocesa v času njegovega zaključka (tj. posnetek podatkov, angl. Snapshot Data). Upravljavec kompenzacij se prične s kompenzacijskim dogodkom tipa *catch*, ki je povezan s kompenzacijsko aktivnostjo preko asociacije. Kompenzacijska aktivnost, ki je lahko ali opravilo (angl. Task) ali podproces ima marker, ki prikazuje, da je podproces uporabljen zgolj za namene kompenzacije in je izven normalnega toka procesa. Kompenzacija je sprožena s strani kompenzacijskega dogodka *throw* (ki ga tipično vrže upravljalec napak), ali rekurzivno s strani drugega upravljavca kompenzacij. Kompenzacijski dogodek je nato zadolžen za specificiranje aktivnosti za katere se bo izvedla kompenzacija. To je lahko storjeno eksplicitno, ali implicitno (aktivnost, ki jo je potrebno kompenzirati je referencirana). Privzeto je kompenzacija sprožena sinhrono, kar pomeni, da kompenzacijski dogodek *throw* čaka na konec izvajanja sproženega upravljavca kompenzacij. Nasprotno je lahko kompenzacija sprožena, ne da bi čakala na konec izvajanja, in sicer tako da nastavimo vrednost atributa kompenzacijskega dogodka *throw* z imenom *waitForCompletion* na *false* [64].

Na podoben način je koncept kompenzacij uporabljen pri upravljanju napak znotraj procesov WS-BPEL. To so aplikacijsko-specifične aktivnosti, ki poskušajo povrniti učinke prejšnjih aktivnosti, ki so bile izvedene kot del večje enote dela, in ga je sedaj

potrebno prekiniti. WS-BPEL ponuja vrsto mehanizmov za kompenzacijo tako, da ponuja zmožnost fleksibilnega nadzora povrnitve. To dosežemo z definiranjem obravnave napak in kompenzacije na aplikacijsko-specifičen način. Zmožnost deklaracije kompenzacijske logike poleg vnaprej-delujoče logike (angl. Forward-Working Logic) predstavlja temelj upravljanja z napakami pri WS-BPEL. Le-to je nadzorovano s strani aplikacije, saj omogoča obsegom (angl. Scopes), da razmejijo del obnašanja, ki je reverzibilen na aplikacijsko-definiran način, in sicer tako da specificirajo upravljavca kompenzacij, pri čemer so lahko obsegi s kompenzacijami in upravljavci napak gnezdeni do poljubne globine [65]. Potrebno je poudariti, da se upravljanje napak razlikuje od upravljanja kompenzacij tako, da do upravljanja napak pride, ko se zgodi napaka znotraj obsega, pri čemer kompenzacija povrne delo uspešno zaključenega obsega.

Kompenzacija se ravno tako razlikuje od transakcijske semantike. Slednja omogoča kombiniranje več operacij v enojno enoto dela, kar daje delovnim tokovom možnost povrnitve vseh sprememb znotraj transakcije v primeru, da pride do napake v katerem-koli delu transakcijskega procesa. Kljub temu transakcije niso vedno primerna rešitev, še posebej pri dolgotrajnih procesih (angl. Long-Running Processes) [192]. Primer: storitev oblaka je implementirana v obliki delovnega toka. Koraki delovnega toka so lahko sestavljeni iz zahteve po določeni storitvi oblaka, čakanja na odobritev manager-ja, oskrbovanja storitve, ter nazadnje plačevanje storitve. Takšen proces bi lahko vzel več dni, pri čemer ni praktično, da bi koraki zahteve, oskrbovanja in plačevanja za storitev participirali znotraj iste transakcije. V takšnih scenarijih je kompenzacija lahko uporabljena za razveljavitev korakov oskrbovanja delovnega toka, v primeru, da pride do kasnejše napake pri procesiranju. Drug primer je, ko oblikujemo tehnični delovni tok za avtomatizirano oskrbovanje virtualnih strojev. V tem primeru se morajo zgoditi številne zaporedne in neodvisne aktivnosti. Pred konfiguracijo virtualnega omrežja posameznega virtualnega stroja mora biti dodeljena in konfigurirana primerna diskovna slika za ta določen virtualni stroj. V primeru omrežne napake mora biti sprožena kompenzacijska aktivnost, ki bo izbrisala diskovne kapacitete ter zagotovila primerno povrnitev oskrbovanega VM-ja.

Trenutno je opazna pomanjkljivost deficitne podpore kompenzacije orkestratorjev oblaka. Omogočena podpora bi prišla prav v številnih poslovnih in tehničnih scenarijih delovnih tokov. Vpeljava podpore bi oblikovalcem in arhitektom delovnih tokov omogočila razvoj delovnih tokov, ki bi bili bolj odporni na napake, in to na učinkovit način. Vendar trenutni orkestratorji oblaka ne podpirajo kompenzacijskih elementov



Slika 5.1

Arhitektura platforme orkestratorja oblaka.

(tj. upravljavci kompenzacij in kompenzacijski dogodki tipa *throw*), kar pomeni, da moramo le-te implementirati ročno.

5.3 Platforme za orkestracijo oblaka

Trenutni orkestratorji oblaka so običajno sestavljeni iz treh ločenih slojev: (1) sloj vmesnika, ki ga sestavlja konzola orkestratorja (tj. grafično okolje "drag-and-drop" za oblikovanje delovnih tokov) in spletne storitve, (2) upravljavski sloj orkestratorja, ki ponuja običajne funkcionalnosti vsake orkestracijske platforme ter (3) podatkovna baza orkestratorja, ki je zadolžena za hrambo relevantnih informacij, kot so procesi, stanja delovnih tokov in različne konfiguracijske informacije. Podatkovna baza orkestratorja prav tako vsebuje vse postavljene delovne tokove, status izvajajočih se delovnih tokov, log datoteke itn. Najbolj relevanten sloj je upravljanje orkestratorja, ki vključuje pogon delovnih tokov (angl. Workflow Engine), knjižnico delovnih tokov (angl. Workflow Library), sistem za verzioniranje (angl. Versioning System), pogon politik (angl. Policy Engine), pogon skript (angl. Scripting Engine), komponento varnosti, integracijske pakete/adapterje (angl. Integration Packs/Adapters) ter sistem za upravljanje napak (angl. Fault Handling System).

Pogon delovnih tokov omogoča izvajanje tehničnih in poslovnih procesov in je v večini primerov realiziran v obliki dveh ločenih pogonov (poslovnega in tehničnega), ki delujeta usklajeno in kolaborativno, da bi dosegli avtomatizacijo storitev v oblaku od

začetka do konca (angl. End-To-End). Da bi ustvarili takšno avtomatizacijo procesov znotraj delovnega toka, mora pogon delovnih tokov podpirati:

1. delovne tokove in objekte, ki jih podpira orkestrator (tj. knjižnico delovnih tokov),
2. poljubne gradnike, ki jih ustvari stranka,
3. objekte in integracijske pakete (včasih poimenovani kot adapterji [60] ali vtičniki [59]), ki jih uvozimo v orkestracijsko platformo oblaka.

Knjižnica delovnih tokov je katalog razširljivih delovnih tokov za kreiranje in izvajanje avtomatiziranih, konfigurabilnih procesov za upravljanje infrastrukture oblaka, kot tudi tretjih (angl. Third-Party) tehnologij [68]. Knjižnica delovnih tokov je v interakciji s podatkovno bazo orkestratorja, saj hrani vse relevantne podatke delovnih tokov. Sistem za verzioniranje pooblašča objekte orkestracijske platforme z asociirano zgodovino verzij. Ta zmožnost omogoča osnovno upravljanje s spremembami (angl. Change Management), ko orkestracijska platforma distribuira procese med različnimi fazami projekta ali različnimi lokacijami. Pogon politik omogoča spremljanje in generiranje dogodkov, da bi lahko reagiral na spreminjajoče se pogoje v orkestracijski platformi. Politike lahko agregirajo dogodke iz platforme ali iz kateregakoli integracijskega paketa, kar omogoča upravljanje spreminjajočih se pogojev na poljubni integracijski tehnologiji. Skriptni pogon ponuja način za kreiranje novih gradnikov za orkestracijsko platformo (tj. integracijske pakete/adapterje) in je v večini primerov obogaten z osnovnim nadzorom verzij (angl. Version Control), preverjanjem tipov spremenljivk (angl. Variable Type Checking), upravljanjem imenskega prostora (angl. Name Space Management) ter upravljanjem izjem (angl. Exception Handling). Pogon skript omogoča razširjanje knjižnic aktivnosti izven kolekcije standardnih aktivnosti in integracijskih paketov/adapterjev. Iz tega razloga so znotraj obstoječih orkestracijskih platform običajno na voljo orkestracijska orodja v obliki SDK-jev (Software Development Kits), ki omogočajo razvijalcem implementacijo in oblikovanje lastnih aktivnosti ter vključevanje le-teh v orkestracijsko platformo.

Komponenta varnosti ponuja napredne varnostne funkcionalnosti, kot so enkriptirana komunikacija SSL (Secure Sockets Layer) med namiznim odjemalcem in strežnikom, upravljanje s pravicami dostopa za ponujanje nadzora nad procesi in objekti ter infrastrukturo javnih ključev (angl. Public Key Infrastructure) za podpisovanje in

enkriptiranje vsebine, uvožene in izvožene med strežniki. Integracijski paketi in adapterji predstavljajo kolekcijo kostumiziranih aktivnosti, ki so specifične za produkt oz. tehnologijo. Kot taki ponujajo podjetjem integracijske pakete z vnaprej definiranimi aktivnostmi za interakcijo z njihovimi sistemi iz orkestracijske platforme. Integracijski paketi omogočajo dostop in nadzor do eksternih tehnologij in aplikacij. Na tak način lahko integriramo objekte in funkcionalnosti delovnih tokov, ki dostopajo do objektov in funkcionalnosti teh eksternih tehnologij (npr. vmesniki za oddaljen nadzor, upravljavska orodja, storitve LDAP, zunanji sistemi za zaračunavanje, podatkovne baze ali e-poštni sistemi).

Upravljanje napak v orkestratorjih oblaka se nanaša na upravljanje izjem na asinhroni način. Kljub temu obstoječe orkestracijske platforme ponujajo zelo osnoven in omejen mehanizem za obravnavo napak, ki jim manjka fleksibilnost pri oblikovanju robustnih delovnih tokov, tolerantnih na napake. Primer: vCenter Orhcestrator [59] podpira obravnavo izjem, ki ujame vse napake, prožene ob zagonu delovnega toka. Vse aktivnosti v delovnem toku, razen odločitvenih, začetnih in končnih aktivnosti, vsebujejo specifične izhodne tipe parametrov, ki služijo zgolj za obravnavo napak. V primeru, da aktivnost naleti na napako med njenim izvajanjem, lahko pošlje signal k upravljavcu izjem, ki prestreže napako in ustrezno reagira na prejete napake. Če upravljavci izjem ne morejo obravnavati določene napake ročno, je mogoče povezati izhodni parameter aktivnosti, ki je tipa izjema, z aktivnostjo izjeme (angl. Exception Activity), ki zaključi izvajanje delovnega toka z neuspelim stanjem. Še več, aktivnosti lahko nastavljajo vezave (angl. Bindings), ki definirajo obnašanje delovnega toka v primeru, da le-ta naleti na napako v aktivnosti. Skratka, vCenter Orhcestrator kot tudi druge orkestracijske platforme, zahtevajo od oblikovalcev delovnih tokov ročno implementacijo upravljavcev izjem in skript. Zaradi tega je pomembno ponujati zanesljivo podporo za obravnavo napak, da se lahko oblikovalci delovnih tokov osredotočajo bolj na implementacijo poslovne logike delovnih tokov in manj na ročno implementacijo robustnih mehanizmov za obravnavo napak. Eden izmed načinov za dosego tega je ločevanje kode (ki se nanaša na upravljanje napak) od logike delovnih tokov.

Na podlagi literarnega pregleda najpomembnejših orkestracijskih platform oblaka [56, 59–63], raziskovanja trenutnih in prihodnjih tehnoloških trendov orkestracije oblaka [193–195] ter testiranja individualnega orkestracijskega sistema v številnih realnih projektih, identificiramo pomembne pomanjkljivosti trenutnih orkestratorjev oblaka: (1) pomanjkanje podpore za upravljanje kompenzacij ter (2) pomanjkanje

podpore za storitve interoperabilnosti. Celovita arhitektura orkestratorjev oblaka je prikazana na sliki 5.1 in izpostavlja obe pomanjkljivosti. Storitev za interoperabilnost abstrahira razlike med več API-ji ponudnikov oblaka tako, da omogoča orkestratorjem zahtevati podatke in računsko moč iz različnih sistemov oblaka preko skupnega vmesnika. Uporablja portabilno abstrakcijo funkcionalnosti, ki so specifične za oblak in omogoča portabilnost procesnih aplikacij oz. aplikacij, ki temeljijo na delovnih tokovih med več oblaki. Kljub temu je aspekt portabilnosti izven obsega te disertacije, saj se osredotočamo izključno na vpeljavo upravljanja kompenzacij. Prepričani smo, da naslavljanje teh izzivov v raziskovalni sferi, kot tudi industriji, predstavlja velik potencial pri prenosu orkestracije oblaka na povsem novo raven.

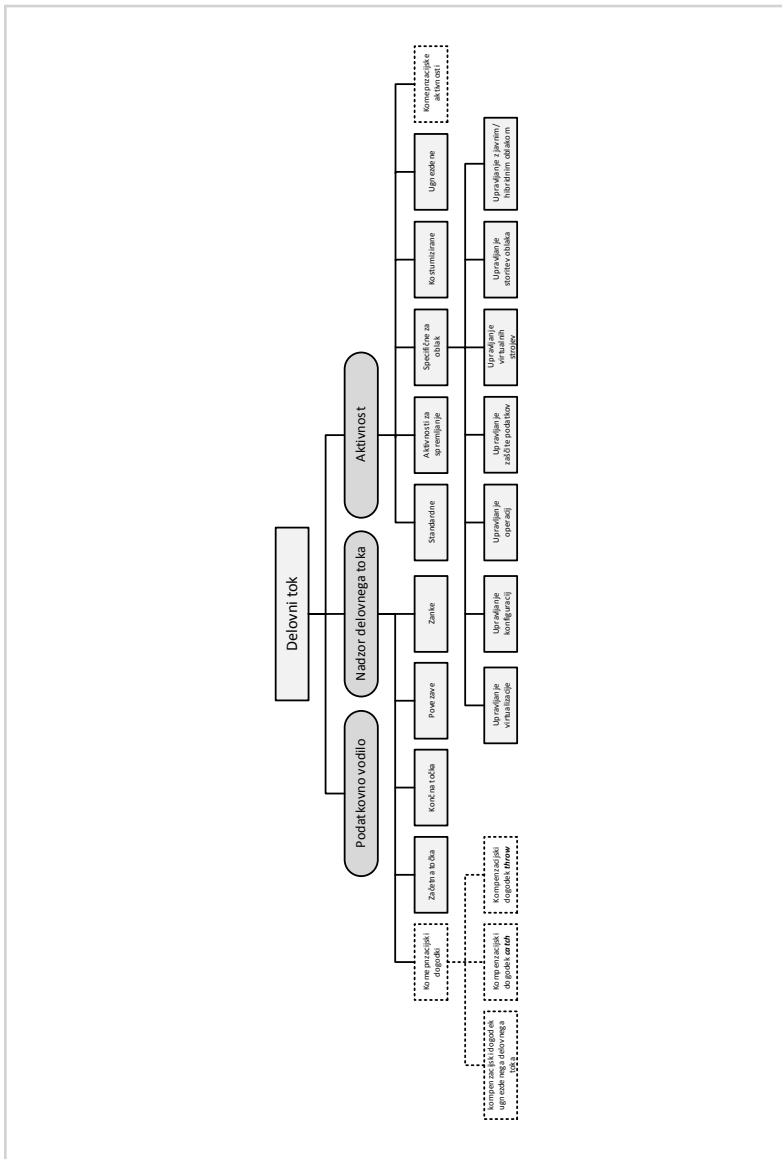
5.4 Vpeljava kompenzacij v orkestracijske platforme oblaka

Cilj tega poglavja je izvesti arhitekturno analizo obstoječih sistemov za delovne tokove, specifične za oblak, in definirati generičen pristop za upravljanje kompenzacij, ki je aplikabilen nad orkestratorji oblaka.

5.4.1 Arhitekturna analiza

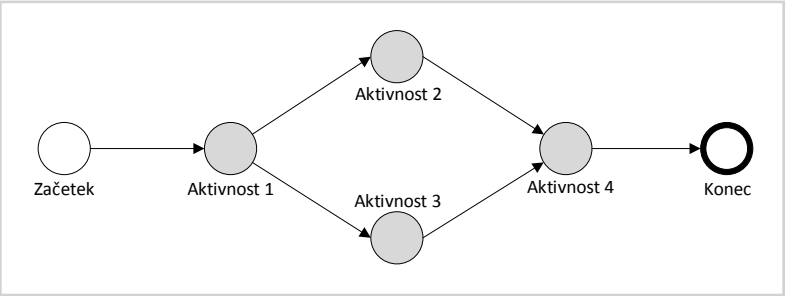
Od leta 1980 je informacijska tehnologija ponujala široko paleto aplikacij, ki podpirajo avtomatizacijo in vodenje poslovnih procesov, pri čemer so sistemi WfMS (Workflow Management Systems) najbolj razvite aplikacije tega tipa [196]. Takšni sistemi popolnoma definirajo, upravljajo in izvajajo delovne tokove preko izvajanja programske opreme, pri kateri je vrstni red izvajanja voden od računalniške reprezentacije logike delovnih tokov [197]. Slika 5.2 prikazuje kategorizacijo elementov orkestracijskih delovnih tokov. Elementi, ki so prikazani s črtkanimi robovi, predstavljajo pomanjkljive konstrukte upravljanja kompenzacij, ki so potrebni za obogatitev orkestratorjev oblaka s to paradigmo.

Znotraj orkestratorjev oblaka predstavljajo delovni tokovi niz aktivnosti, ki uporabljajo podatke, izvajajo opravila in objavljajo podatke za druge aktivnost v delovnem toku [56, 59–63]. Običajno vsebujejo inštrukcije za avtomatizirano opravilo ali proces. To je lahko na primer proces za avtomatizacijo opravil, ki temelji na virih, ali opravilo za procesiranje podatkov znotraj oblaka. Individualne korake v celotnem delovnem toku označujemo kot aktivnosti. Znotraj delovnega toka dodatni nadzorni elementi ponujajo informacije in inštrukcije za nadzor zaporedja aktivnosti. Te lastnosti lahko



Slika 5.2

Kategorizacija elementov orkestracijskih delovnih tokov oblaka.



Slika 5.3

Konceptualna arhitektura delovnega toka s štirimi aktivnostmi.

modificiramo za konfiguracijo obnašanja delovnih tokov [198]. Delovni tokovi uporabljajo široko paleto kostumizacijskih možnosti, tj. lastnosti delovnih tokov (angl. Workflow Properties) in pravic delovnih tokov (angl. Workflow Permissions). Lastnosti delovnih tokov ponujajo možnost konfiguracije individualnega delovnega toka (npr. ime in opis delovnega toka). Pravice delovnih tokov opisujejo dovoljenja za dostop do delovnih tokov. Dodatnim uporabnikom lahko na primer dodelimo pravice za vpogled in spreminjanje delovnih tokov. Konceptualna arhitektura delovnega toka s štirimi aktivnostmi je prikazana na sliki 5.3.

Aktivnosti

Aktivnosti so osnovni gradniki delovnega toka. Vse aktivnosti, ne glede na specifično tehnologijo, si delijo skupno obnašanje. V splošnem individualne aktivnosti izvajajo sledeče akcije: (1) dostopajo do objavljenih podatkov, (2) izvajajo akcije tako, da izvajajo poljubne skripte (npr. skripte v obliki API-jev, ki so specifični za oblak) ter (3) objavljajo nove podatke tako, da naslednje zaporedne aktivnosti lahko do njih dostopajo. Aktivnost je koračni proces, ki je lahko ali atomarna (skriptno opravilo) ali razgradljiva (gnezdene aktivnosti), in je izvajana s strani sistema (avtomatizirana aktivnost) ali s strani človeka (ročna aktivnost). Vse aktivnosti si delijo skupne atribute in obnašanje, kot so na primer stanja in prehodi stanj, ter so običajno organizirane v kategorije, kot so standardne aktivnosti, aktivnosti za spremljanje, aktivnosti specifične za oblak, kostumizirane aktivnosti ter vgrajene aktivnosti (slika 5.2).

Standardne aktivnosti so atomarne aktivnosti, ki vključujejo osnovni nabor aktivnosti za oblikovanje delovnih tokov. Običajno vključujejo aktivnost za podporo pošiljanju e-poštnih notifikacij, aktivnost za podporo delu in manipulaciji podatkov znotraj

delovnih tokov, aktivnost, ki je zadolžena za izvajanje sistemskih ukazov, kot so izvajanje progama, aktivnost, ki reagira na sistemske dogodke itn. Aktivnosti za spremljanje vključujejo nabor aktivnosti, ki so sprožene od stanja ali dogodka opravila izven delovnega toka. Primer: aktivnost za spremljanje lahko preveri status naslova IP določenega virtualnega stroja ali čaka, da se pojavi določen dogodek v dogodkovnem logu (angl. Event Log). Takšne aktivnosti so običajno zahtevane kot začetni pogoji znotraj delovnega toka in ne morejo biti sprožene s strani drugih aktivnosti delovnega toka. Aktivnosti, ki so specifične za oblak, vključujejo nabor aktivnosti, ki so povezane z upravljanjem oblaka in so lahko kategorizirane v sedem skupin. Upravljanje virtualizacije, upravljanje konfiguracij, upravljanje operacij, upravljanje zaščite podatkov, upravljanje virtualnih strojev, upravljanje storitev in upravljanje javnega/hibridnega oblaka.

Kostumizirane aktivnosti razširjajo standardne aktivnosti. Pravzaprav so te aktivnosti izdelane po meri in so prilagojene vsaki skupini uporabnikov, da zadovoljijo njihove specifične/posebne zahteve. Takšen tip aktivnosti pride prav v primeru, da funkcionalnost, ki jo zahtevamo, ni prisotna znotraj orkestracijske platforme. V ta namen večina orkestratorjev oblaka ponuja orkestracijska orodja v obliki SDK-jev. Z uporabo SDK-jev lahko razvijalci implementirajo in oblikujejo lastne aktivnosti (aktivnosti po meri) in jih vključijo v orkestracijsko platformo. Gnezdene aktivnosti so zadolžene za klicanje delovnih tokov, ki jih specificiramo. Podatke lahko v delovnih tokovih prenašamo tako, da konfigurirano začetno točko v klicanem delovnem toku in vrnemo podatke iz klicanega delovnega toka tako, da konfiguriramo končno točko (začetna in končna točka sta podrobneje razloženi v nadaljevanju). Gnezdena aktivnost je lahko uporabljena za klicanje generičnih delovnih tokov, ki izvajajo zgolj specifične akcije, in so neodvisne od tega, kako je delovni tok klican. Kreiramo na primer delovni tok, ki kliče ločen delovni tok za izvedbo procedure za vzdrževanje varnostne kopije. Ta procedura nato kliče delovni tok, ki zaustavi storitev, drug delovni tok, ki naredi varnostno kopijo podatkov, in nato delovni tok, ki ponovno zažene storitev.

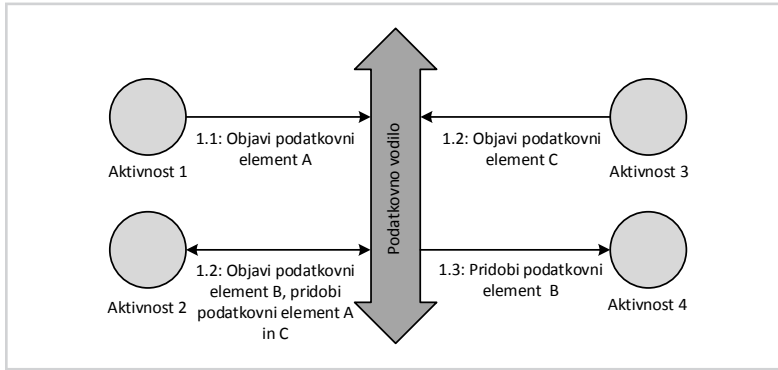
Nadzor delovnega toka

Nadzor delovnega toka običajno ponuja začetno točko (angl. Starting Point), končno točko (angl. Ending Point), povezave (angl. Links) in zanke (angl. Loops). Vsak delovni tok ima lahko samo eno začetno točko. Začetna točka je aktivnost, ki se samodejno izvede ob začetku delovnega toka, medtem ko končna točka označuje konec

avtomatiziranega delovnega toka. Vsaka aktivnost v delovnem toku se izvede za prejšnjo aktivnostjo in to počne tako dolgo, dokler delovni tok ni zaključen (na zaporedni ali paralelni način). Povezave so zadolžene za povezovanje individualnih aktivnosti v delovnem toku. Povezave kličejo naslednje aktivnosti v delovnem toku takoj za tem, ko se prejšnja aktivnost uspešno zaključi in v mnogih primerih ponuja zmožnost filtriranja podatkov tako, da lahko omejimo prenos podatkov do poznejših aktivnosti znotraj delovnega toka. To je doseženo z uporabo pogojev povezave (angl. Link Conditions), kjer lahko izdelamo možnost razmejevanja znotraj delovnih tokov. Primer: delovni tok mora zaustaviti aplikacijski strežnik preden zanj izdela varnostno kopijo. V primeru, da se strežnik s podatkovno bazo zaustavi pravilno, delovni tok zažene opravilo izvajanja varnostne kopije. Če se podatkovna baza ne zaustavi pravilno, je poslana e-pošta administratorju, da bi eskalirali težavo. Zanka na drugi strani omogoča (1) gradnjo samodejnih ponovitev/zank, (2) gradnjo čakajočih pogojev (angl. Wait Conditions) ter (3) spremljanje vseh aktivnosti znotraj delovnega toka. Vsaka aktivnost lahko za namene validacije kreira zanko, da bi lahko ponovili operacijo v primeru napake ali testirali izhodne operacije aktivnosti. Za doseganje obogatitev z upravljanjem kompenzacij predlagamo dodatne elemente delovnega toka: kompenzacijsko aktivnost (angl. Compensation Activity), ki je vključena v vejo aktivnosti, ter tri kompenzacijske dogodke: kompenzacijski dogodek gnezdenega delovnega toka (angl. Compensation Event Subworkflow), kompenzacijski dogodek *catch* (angl. Catch Compensation Event) ter kompenzacijski dogodek *throw* (angl. Throw Compensation Event). Le-ti razširjajo vejo nadzora delovnih tokov. Vsak izmed elementov je podrobneje opisan v poglavju 5.4.2.

Podatkovno vodilo

Podatkovno vodilo (angl. Data Bus) je mehanizem, ki posreduje informacije iz ene aktivnosti delovnega toka na drugo aktivnost. Elementi povezave uporabljajo to informacijo za dodajanje odločitvenih zmožnosti delovnim tokovom. Za vsako aktivnost je mogoče določiti podatkovne elemente, za katere želimo, da jih ta aktivnost objavi. Vsak objavljen podatkovni element, ki ga specificiramo, je razpoložljiv na podatkovnem vodilu in teče po poti delovnega toka. Vsak delovni tok se izvaja znotraj lastnega procesa, pri čemer znotraj vsakega takšnega procesa obstaja tudi podatkovno vodilo. Ko se delovni tok preneha izvajati, so podatki, objavljeni v podatkovnem vodilu, izgubljeni. Število podatkovnih elementov, ki jih producirajo aktivnosti, kakor tudi konfiguracije



Slika 5.4

Primer prenosa sporočil med aktivnostmi znotraj delovnega toka.

lastnosti, lahko vplivajo na število, kolikokrat se sme aktivnost izvesti in koliko podatkovnih elementov je lahko prisotnih na izhodu aktivnosti. Slika 5.4 prikazuje primer prenašanja sporočil med štirimi aktivnostmi znotraj istega delovnega toka. V tem primeru aktivnost 1 objavi podatkovni element A v podatkovno vodilo. Primer: aktivnost 1 pridobi sporočilo SOAP iz spletne storitve, ki je objavljena na podatkovnem vodilu. Naslednja aktivnost 3 objavi podatkovni element C (npr. aktivnost 3 pridobi dodatno sporočilo SOAP in ga objavi na podatkovno vodilo), medtem ko v istem času paralelna aktivnost 2 objavi podatkovni element B takoj za tem, ko je le-ta pridobila podatkovni element A in C. Primer: aktivnost 2 je konfigurirana za naročanje na prej pridobljeno sporočilo SOAP in izvede poizvedbo za pridobivanje določenih podatkov znotraj teh sporočil in nato objavi agregirane vrednosti na podatkovno vodilo. Na koncu aktivnost 4 pridobi agregiran podatkovni element B iz podatkovnega vodila in ga vrne v obliki rezultatov delovnega toka.

Primer orkestracije delovnih tokov za oblak

V orkestriranem okolju razvijalec vnese želene specifikacije za zahtevano storitev oblaka (preko portala, spletne storitve orkestratorja, ali katerega drugega vmesnika). Razvijalec pri tem vnese številna polja, ki definirajo fizične/virtualne vire, hitrost CPU-ja, diskovni prostor, pomnilnik, storitvene nivoje itn. Sistem nato:

1. verificira ali so vsa potrebna soglasja/odobritve na mestu, po možnosti pošlje opozorila za dodatne odobritve k ustreznim uporabnikom. Ker gre za avtoma-

tiziran proces s ključnimi nadzornimi parametri, so te zahteve v večini primerov vnaprej odobrene, vendar morajo biti še vedno označene kot preverjene/odobrene zaradi skladnosti z regulativami in predpisi (v tem primeru predvidevamo, da ima končni uporabnik zadostna sredstva na svojem računu, da bi financiral izdelavo storitve),

2. validira zadostne kapacitete za procesiranje zahteve, medtem ko najprej identificira SLA-je stranke in okolje, v katero bo njena storitev oblaka postavljena, ter nato preišče okolje, da bi determiniral lokacijo oskrbovanja storitve,

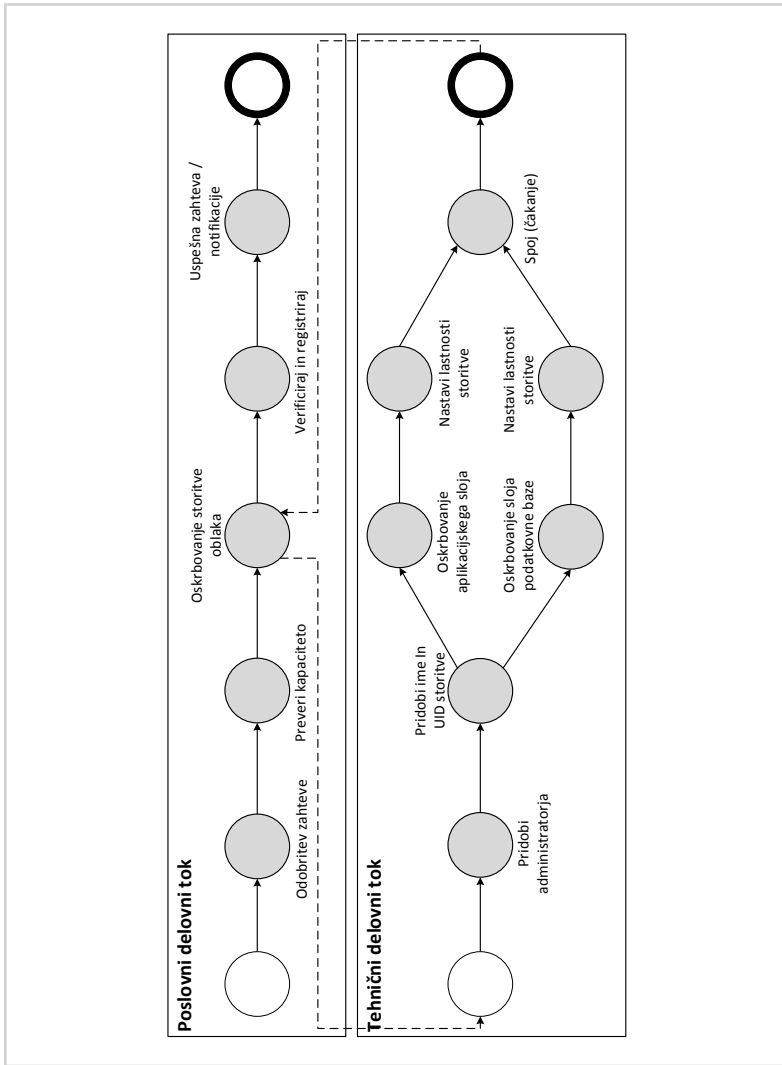
3. izvede tehnični delovni tok preko številnih skript, da bi oskrbel vire, ki jih zahteva storitev (npr. shramba, omrežje, računska moč itn.),

4. naredi poizvedbo po novo ustvarjenih storitvah in verificira, ali ustreza zahtevi uporabnika oz. stranke, validira, ali stranka dobi storitev, za katero plača, ter registrira novo ustvarjeno sredstvo v inventorij (npr. Configuration Management Database – CMDB),

5. obvesti vse stranke o uspešnem zaključku.

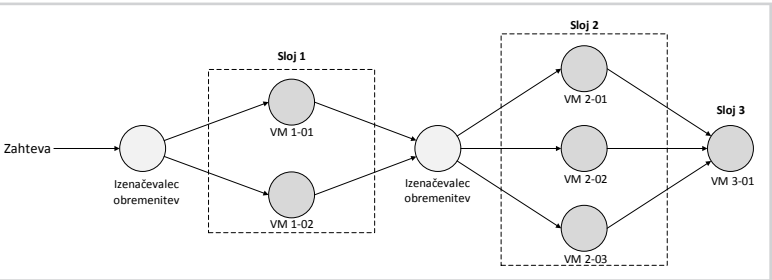
Opisano izpolnjevanje storitvene zahteve (angl. Service Request Fulfillment) omogoča avtomatizacijo storitev oblaka od začetka do konca, medtem ko uporablja tako poslovne kot tudi tehnične orkestracijske tehnike. Primer takšne dvoslojne orkestracije, ki uporablja delovne tokove preko poslovne in tehnične domene, je prikazan na sliki 5.5. V primeru napake znotraj aktivnosti spoja (kot del gnezdenega delovnega toka "Oskrbovanje storitve oblaka"), morajo biti zaradi prekoračenega čakalnega časa (specificirana/nastavljena časovna omejitev) za oskrbovanje vseh vključenih storitvenih slojev, vse že uspešno zaključene aktivnosti razveljavljene in ustrezno kompenzirane. V primeru večslojne storitvene arhitekture so določena vozlišča običajno odvisna od drugih vozlišč storitve in zato zahtevajo posebno pozornost pri oblikovanju uvedbe upravljanja kompenzacij.

Večslojna arhitektura ponuja fleksibilen, modularen pristop za oblikovanje aplikacij, specifičnih za oblak. Vsak aplikacijski sloj ponuja določene funkcionalnosti svojemu predhodnemu sloju in uporablja funkcionalnosti ponujene s strani naslednika, da bi izvršil svojo vlogo v celotnem procesiranju zahteve. Slika 5.6 prikazuje primer trislojne aplikacije, kjer nova zahteva uporabnika vstopi v omrežje pri prvem sloju odprtega



Slika 5.5

Primer dvoslojne interakcije delovnih tokov.



Slika 5.6
Primer trislojne aplikacije.

sporočilnega omrežja m-slojne aplikacije. Ko so zahteve enkrat procesirane, so takoj posredovane naslednjim slojem in končno zapustijo aplikacijo iz zadnjega sloja, za tem ko so preiskale vseh m-slojev. Omenjen pristop rezultira v številne postavitvene odvisnosti (angl. Deployment Dependencies) med storitvenimi sloji. Primer: vozlišča v sloju 1 (npr. spletni sloj) so odvisna od izenačevalca obremenitev drugega sloja in posledično od vozlišč v sloju 2 (npr. aplikacijski sloj), saj zahtevajo dostop do poslovne logike aplikacije. Vozlišča iz sloja 2 so odvisna od sloja 3 (npr. podatkovne baze), saj zahtevajo povezljivost do podatkovne baze preko naslova IP. Iz tega razloga mora biti med konstruiranjem storitve oblaka upoštevan vrstni red postavitve aplikacije (termina "aplikacija, specifična za oblak" ter "storitev oblaka" sta v disertaciji uporabljena izmenjujoče in imata enak pomen). Ko so storitveni sloji postavljeni, moramo kot posledico prav tako upoštevati odvisnosti. Primer: avtomatizacija konstrukcije storitvenih slojev vključuje tri korake. V prvem koraku (korak 1) orkestrator generira sliko virtualnega stroja (npr. Tomcat), glede na parametre specificirane na portalu oblaka (npr. stroškovni model, cena, omejitve replikacije, CPU, pomnilnik, shramba, tip operacijskega sistema itn.). V koraku 2 je VM najprej zagnan, nato se izvršijo samodejno-izvajajoče se skripte (angl. Auto-Running Scripts) v tem stroju, poleg tega se uporabijo dodatni parametri (lahko so tudi specificirani na portalu, npr. uporabniško ime, geslo, številka vrat), ki so namenjeni konfiguraciji strežnika Tomcat. V koraku 3 je Tomcat asociiran s svojim vhodnim izenačevalnikom obremenitve (tj. izenačevalec obremenitve zazna in registrira Tomcat VM), ki distribuira delovne obremenitve med več vozlišči drugega sloja. Dekonstrukcija tega sloja mora slediti obratnim korakom. Pri tem je v koraku 1 Tomcat VM ločen od svojega vhodnega izenačevalca obremenitve, v koraku 2 se VM zaustavi, medtem je v koraku 3 izvedena dekonstrukcija VM-ja. Poleg analize odvisno-

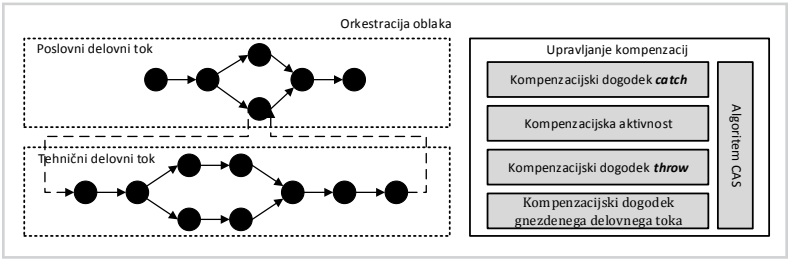
sti, morajo orkestratorji oblaka upoštevati tudi pojav delnega izvajanja (angl. Partial Execution). Primer: če je sloj podatkovne baze uporabljen še s strani drugih aplikacij v sistemu, instance tega sloja ne morejo biti uničene kot del kompenzacije. V ta namen mora orkestrator najprej določiti vse možne odvisnosti, ki jih ima ta podatkovna baza z ostalimi storitvami računalniškega oblaka. V primeru, da ni nobenih odvisnosti, lahko kompenzacijska aktivnost terminira instance sloja, v nasprotnem primeru pa kompenzacijska aktivnost ne bo izvršena in ne bo izvedla specificiranih operacij povrnitve.

Doslej je bilo upravljanje kompenzacij izvedeno ročno z implementacijo razveljavitvenih operacij že zaključenih aktivnosti v eni izmed prihajajočih aktivnosti ali znotraj upravljavca napak. Dogodkovno usmerjen mehanizem, ki je odporen na napake, in definira, kako kompenzirati individualne ali kompozitne aktivnosti znotraj delovnega toka, v primeru izjem oz. napak, še ni bil naslovljen in bi predstavljal koristno ter inovativno funkcionalnost orkestratorjev oblaka. Trenutno noben izmed slojev orkestracijske arhitekture ne ponuja podpore za upravljanje kompenzacij in kompenzacijskih dogodkov. Iz tega razloga predlagamo obogatitev orkestracijske platforme oblaka z upravljanjem kompenzacij, ki naslavljajo prej omenjene izzive in pomanjkljivosti.

5.4.2 Predlagana rešitev za upravljanje kompenzacij

Ker storitev oblaka vključuje delovne tokove v številnih tehničnih in poslovnih domenah, naš predlog razširja arhitekturo orkestratorjev oblaka s principi upravljanja kompenzacij tako za tehnične, kot tudi poslovne orkestracijske delovne tokove. Poleg dostopanja do objavljenih podatkov, izvajanja akcij z izvedbo poljubnih skript in objavljanja novih podatkov za prihajajoče aktivnosti, mora vsaka aktivnost delovnega toka znotraj orkestratorja ponujati podporo za naslednje konstrukte (slika 5.7):

- upravljavca kompenzacij (kompenzacijski dogodek *catch* ter kompenzacijsko aktivnost),
- kompenzacijski dogodek *throw*,
- kompenzacijski dogodek gnezdenega delovnega toka,
- algoritem CAS (Compensation Activity Search).



Slika 5.7

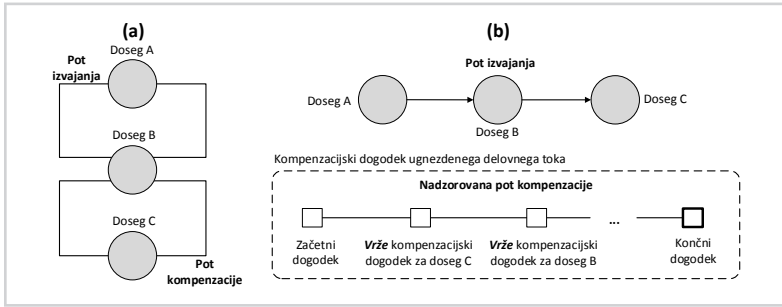
Obogatitev orkestratorjev oblaka s konstrukti upravljanja kompenzacij.

Upravljavec kompenzacij

Upravljavec kompenzacij je sestavljen iz dveh delov: kompenzacijskega dogodka *catch* ter kompenzacijske aktivnosti, ki je povezana s kompenzacijskim dogodkom *catch* preko asociacije. Kompenzacijski dogodek *catch* je lahko ali mejni dogodek (angl. Boundary Event), ki je pripet h kompenzacijski aktivnosti, ali upravljavčev začetni dogodek (v primeru kompenzacijskega dogodka gnezdenega delovnega toka). Kompenzacijski mejni dogodek je aktiviran, ko se aktivnost, h kateri je pripet, uspešno zaključi. V tem trenutku se ustvari ustrezna naročnina na kompenzacijske dogodke in je odstranjena šele, ko je sprožen kompenzacijski dogodek ali ko se zaključi začetna instanca delovnega toka. Na drugi strani je kompenzacijski dogodek gnezdenega delovnega toka zadolžen za upravljanje dogodka znotraj konteksta danega gnezdenega delovnega toka ali delovnega toka, in se zmeraj začne z začetnim dogodkom, ki mu sledi zaporedje tokov. Takšen tip dogodka pravzaprav ustvari obseg in je instanciran kot gnezden delovni tok, vendar ni instanciran s strani običajnega nadzornega toka, ampak zgolj ob proženju asociiranega začetnega dogodka. Kompenzacijska aktivnost je lahko običajna aktivnost (atomarno opravilo) ali gnezdena aktivnost in je izven normalnega toka delovnega toka. V vsakem primeru je zadolžena za izvedbo povrnitvene operacije že zaključenih aktivnosti in izvedbo ustrezne kompenzacijske logike.

Kompenzacijski dogodek throw

V nadaljevanju definiramo kompenzacijski dogodek *throw*, ki nam omogoča proženje upravljavca kompenzacij za doseg, v katerem se le-ta nahaja, ali za specifično aktivnost iz tega dosega. Ta tip dogodka je zadolžen za specificiranje/referenciranje aktivnosti ali dosega, za katerega se bo izvedla kompenzacija. Če dogodek identificira aktivnost, potem je to tista aktivnost, ki bo kompenzirana. V nasprotnem primeru je kompenzacija



Slika 5.8

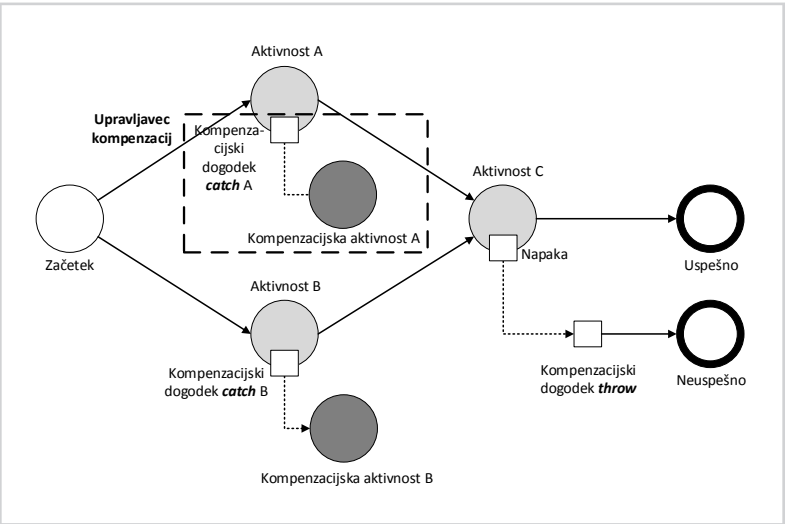
(a) Obmejna obravnava dogodkov. (b) Notranja obravnava dogodkov.

prenesena na vse aktivnosti, ki so se zaključile znotraj instance delovnega toka, vključno s korenskim delovnim tokom (angl. Top-Level Workflow) in vsemi gnezdenimi delovnimi tokovi. Kompenzacija je lahko prožena sinhrono, tako da kompenzacijski dogodek *throw* čaka na zaključek proženega upravljavca kompenzacij, ali asinhrono, pri čemer je lahko kompenzacija prožena, ne da bi čakala na njen zaključek. V primeru, da ima instanca večinstančne karakteristike (angl. Multiple Instance Characteristics), je poklican upravljavec kompenzacij za vsako instanco, ki se uspešno zaključi. Primer: več instanc tipično obstaja za gnezdene delovne tokove v zankah (angl. Loop Subworkflows) ali za večinstančne gnezdene delovne tokove (angl. Multi-Instance Subworkflows). Vsaka izmed omenjenih ima svojo instanco kompenzacijskega dogodka gnezdenega delovnega toka, ki ima dostop do specifičnih podatkovnih posnetkov narejenih v času zaključka te določene instance. Slednje pomeni, da se v primeru proženja kompenzacije za delovne tokove v zankah ali večinstančne gnezdene delovne tokove, individualno prožijo kompenzacije za vse instance znotraj trenutnega dosega.

Notranja in mejna obravnava dogodkov

Upravljavec kompenzacij mora biti definiran za obseg in lahko ima največ eno kompenzacijsko aktivnost. Kompenzacijska aktivnost je dodatno lahko strukturirana aktivnost (angl. Structured Activity) v obliki zaporedja, ki ima lahko podrejene aktivnosti (angl. Child Activities). Obstajata dva možna pristopa za doseganje upravljanja kompenzacij v orkestratorjih oblaka:

- *Mejna obravnava dogodkov* (angl. Boundary Event Handling) – v tem primeru uporabimo aktivnost *compensate event definition* znotraj upravljavca kompenza-



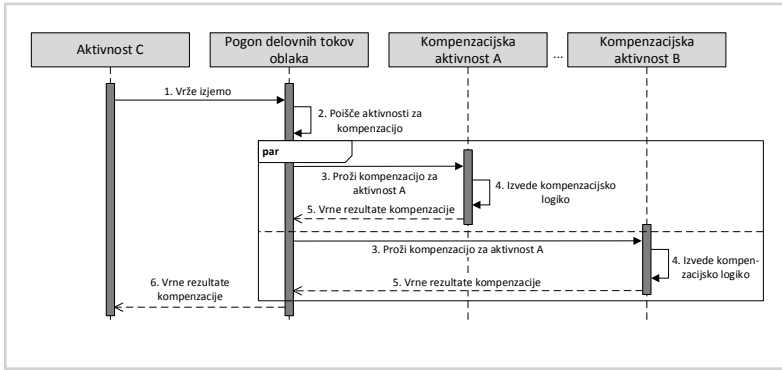
Slika 5.9

Obogatitev aktivnosti delovnih tokov z upravljanjem/obravnavo kompenzacij.

cij ali upravljavca napak, z namenom specificiranja kompenzacije na vseh notranjih dosegih, ki so se že uspešno izvedli, v privzetem vrstnem redu (slika 5.8). V tem primeru je kompenzacija izvedena v obratnem vrstnem redu od izvedbe, kar pomeni, da se aktivnosti, ki so se izvedle zadnje, kompenzirajo prve.

- *Notranja obravnava dogodkov* (angl. Inline Event Handling) – v tem primeru uporabljamo aktivnost *compensate event definition* znotraj upravljavca kompenzacij ali upravljavca napak, da bi specificirali kompenzacijo za imenovan notranji doseg. Ta metoda ponuja nadzor nad vrstnim redom izbire kompenzacijskih aktivnosti v dosegu (slika 5.8).

Naslednji primer delovnega toka prikazuje način deklaracije poljubne aktivnosti delovnega toka, ki je lahko kompenzirana z uporabo upravljavca kompenzacij (slika 5.9). V tem primeru imamo delovni tok s tremi aktivnostmi: aktivnost A, aktivnost B in aktivnost C. Aktivnosti A in B imata pripetega upravljavca kompenzacij. Aktivnost C ima pripetega upravljavca napak, ki je zadolžen za proženje kompenzacijskega dogodka v primeru napake znotraj aktivnosti C.



Slika 5.10

Diagram zaporedja, ki prikazuje interakcijo med udeleženci upravljanja/obravnave kompenzacij.

Interakcija med udeleženci upravljanja kompenzacij je prikazana na sliki 5.10. Če pride do napake med izvajanjem aktivnosti C, se zgodijo naslednji koraki: (1) aktivnost C proži izjemo zaradi pojava napake, (2) pogon delovnih tokov oblaka (angl. Cloud Workflow Engine) pokliče operacijo *find():List<Activity>*, ki je zadolžena za preiskovanje usmerjenega grafa, pri čemer sledi inverznim povezavam (angl. Inversed Edges) in išče aktivnosti, ki morajo biti kompenziran, (3) ko so enkrat detektirani vsi kompenzacijski dogodki *catch*, pogon delovnih tokov proži referencirane kompenzacijske dogodke *catch*, ki so pripeti k aktivnostim A in B, (4) kompenzacijske aktivnosti A in B izvedejo kompenzacijsko logiko v obratnem vrstnem redu izvajanja (aktivnost, ki se zaključi zadnja, se kompenzira prva) ali v specifičnem vrstnem redu – odvisno od tipa obravnave kompenzacijskih dogodkov (mejna ali notranja), in nazadnje (5) vrne rezultate kompenzacije k pogonu delovnih tokov oblaka.

Algoritem za iskanje kompenzacijskih dogodkov catch

Delovni tokovi v orkestratorjih oblaka se lahko obravnavajo kot usmerjeni grafi (ali digrafi). Usmerjen graf ima množico vozlišč, ki jih med seboj povezujejo povezave, pri čemer ima vsaka povezava usmerjenost. Formalno lahko usmerjeni graf zapišemo kot par $G = (V, E)$ v množici V, čigar elementi se imenujejo vozlišča; ter množica urejenih parov vozlišč E, ki jih imenujemo povezave. Povezava $E = (x, y)$ je obravnavana kot usmerjena iz x do y; y se imenuje glava povezave in x se imenuje rep povezave; y je neposredni naslednik od x, in x je neposredni predhodnik od y. Če pot, sestavljena iz ene ali več zaporednih povezav, vodi iz x v y, potem je y naslednik od x, in

```
Input: A graph G
Output: The list of activities to be compensated, or null if no compensation vertices exists in G

1  procedure CAS (G):
2    L ← empty list
3    Q ← empty queue
4    endNodes ← list of all end nodes in G
5    enqueue all endNodes onto Q
6    while Q is non-empty do
7      t ← Q.dequeue()
8      if t has an attached catch compensation event and has completed execution then
9        add activity/node t to list L
10       if t has a nested workflow Gt then
11         Lt ← CAS (Gt)
12         append Lt to L
13     for each edge e in G.incomingEdges (t) do
14       o ← G.predecessorVertex (t, e)
15       if o is not marked as visited then
16         enqueue o onto Q
17         mark t as visited
18 return list L
```

Slika 5.11

Algoritem CAS za iskanje kompenzacijskih dogodkov catch.

x predhodnik od y. Povezava (y, x) se imenuje obratna povezava od povezave (x, y). Najodmevnejši reprezentaciji digrafa sta matrika sosednosti (angl. Adjacency Matrix) ter seznam sosednosti (angl. Adjacency List).

Skladno s tem predlagamo nov algoritem CAS za preiskovanje delovnih tokov orkestratorjev oblaka in iskanje ustreznih aktivnosti za kompenzacijo. Algoritem CAS analizira graf in išče kompenzacijske dogodke *catch*, kadarkoli je prožen kompenzacijski dogodek *throw*, ter sledi pristopu algoritma Breadth-First Search (BFS). Algoritem CAS uporablja podatkovno strukturo vrsta za shrambo vmesnih rezultatov in podatkovno strukturo seznam za shrambo aktivnosti, ki jih je potrebno kompenzirati, ob tem ko preiskuje vozlišča grafa. V osnovi algoritem prične pri končnih vozliščih (enemu ali več) in preiskuje vsa neposredna predhodna vozlišča ter ponavlja proceduro, dokler graf ni v celoti preiskan. Vedite, da v tem primeru končna vozlišča označujejo vozlišča, ki nimajo nobenih naslednikov. Za razliko od algoritma BFS, naš algoritem prične preiskovanje od inicialnega vozlišča (tj. končno vozlišče) do začetnega vozlišča, tako da sledi ”inverznim povezavam”. Preiskovanje algoritma CAS je sestavljeno iz

treh esencialnih operacij: (1) obišče in pregleda vozlišče grafa, (2) pridobi dostop za obisk predhodnih vozlišč trenutno obiskanega vozlišča in (3) poišče vse kompenzacijske dogodke *catch* znotraj danega delovnega toka in jih hrani v seznamu aktivnosti za kompenzacijo. Ker je graf lahko sestavljen iz poljubnega števila ciklov, je potrebno pri pregledu označiti vsa vozlišča kot obiskana. Na tak način se izognemo izvajanju algoritma v neskončni zanki (v primeru cikličnega grafa, angl. Cyclic Graph). V primeru gnezdenega delovnega toka, naš algoritem implementira rekurzivno iskanje kompenzacijskih dogodkov *catch* (vrstica 10-12) do poljubne globine. Psevdokoda algoritma CAS je prikazana v algoritmu na sliki 5.11.

Ko kompenziramo gnezdeno aktivnost, ima izvedba, ki je uporabljena za izvajanje upravljavca, dostop do lokalnih spremenljivk gnezdene aktivnosti v stanju, kot so bile takrat, ko je gnezdena aktivnost zaključila izvajanje. Da bi to dosegli, mora biti narejen posnetek spremenljivk delovnega toka, ki je asociiran z dosegom izvajanja (tj. izvedba, ki je ustvarjena za izvajanje gnezdenega delovnega toka). Za boljše razumevanje dodajam naslednji primer. Če je upravljavec kompenzacij poklican, le-ta vidi zamrznjeno stanje lokalno definiranih spremenljivk dosega, v stanju kot so bile ob zaključku izvajanja dosega. V primeru, da se je doseg izvedel v zanki večkrat, potem vsaka instanca kompenzacije posname vrednosti lokalnih spremenljivk v času zaključka dosega. Kljub temu je upravljanje kompenzacij asociirano z dosegom. Doseg lahko obravnavamo kot logično enoto dela z naborom aktivnosti, ki potrebuje razveljavitev oz. kompenzacijo in morebiti ponovni poskus izvajanja, medtem ko ostali deli delovnega toka normalno tečejo dalje.

5.5 Evalvacija

Kot del evalvacije predlagamo nove razširitve BPMN 2.0 s specifikami oblaka (poglavje 5.5.1), opišemo primer uporabe izvršljivega BPMN 2.0 za namene orkestracije okolij oblaka in vključitve principov upravljanja kompenzacij v arhitekture orkestratorjev oblaka (poglavje 5.5.2), ter verificiramo predlagano rešitev v obliki implementacije PoC (poglavje 5.5.3).

Ker je naš namen adaptirati izvršljivi BPMN 2.0 za orkestracijo virov/storitev oblaka in učinkovito kompenzacijo orkestracijskih delovnih tokov, organiziramo fundamentalne storitve orkestratorjev oblaka (prikazane na sliki 5.1) v dve kategoriji:

Tabela 5.1

Primerjava podpore storitev med obstoječimi orkestratorji oblaka in procesnim pogonom BPMN.

Upravljavske storitve	Obstoječi orkestratorji oblaka	Procesni strežniki BPMN 2.0
Pogon delovnih tokov	+	+
Knjižnica delovnih tokov	+	+
Verzioniranje	+	+
Pogon politik	+	+
Pogon skript	+	+
Varnost	+	+
Integracijski paketi/ adapterji	+	-
Upravljanje napak	+	+
Upravljanje kompenzacij	-	+
Interoperabilnost	-	-

1. nove funkcionalnosti orkestratorja, ki jih moramo vključiti v pogone BPMN 2.0,

2. obstoječe storitve, ki jih lahko neposredno uporabimo iz pogona BPMN 2.0, da bi orkestrirali opravila, specifična za oblak.

Trenutno BPMN ne podpira delovnih tokov s specifikami oblaka (angl. Cloud-Enabled Workflows). Kljub temu je z izvršljivim BPMN 2.0 mogoče identificirati tri različne aplikacijske domene za modeliranje jezikov (čisti opis, simulacija in izvajanje procesov) ter uporabiti ponujen razširitveni mehanizem za dopolnitev pomanjkljivosti jedrnih elementov BPMN. Iz tega razloga zagovarjamo stališče, da je izvršljivi BPMN 2.0 lahko uporabljen kot ena izmed standardnih tehnologij za orkestracijo delovnih tokov, specifičnih za oblak, in doseganje obogatitve orkestratorjev oblaka s principi upravljanja kompenzacij. To je mogoče, ker specifikacija BPMN 2.0 dodaja izvršljivo-usmerjene načrtovalske elemente (angl. Execution-Oriented Design Elements), ki ponujajo teoretično definicijo o tem, kaj naredi model izvršljiv ter definira razširitveni mehanizem za razširitve procesnega modela, kot tudi razširitve grafičnega modela [199]

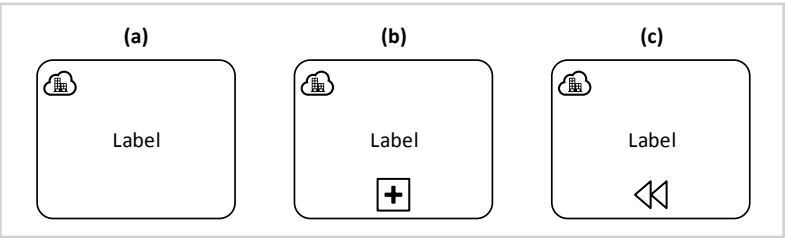
Tabela 5.1 prikazuje primerjavo podpore storitev med obstoječimi orkestratorji oblaka in procesnim pogonom BPMN. Prvi stolpec prikazuje podporo storitev obstoječih orkestratorjev oblaka (označeno s "+/-"). Na podoben način drugi stolpec tabele iz-

postavlja nove storitve, ki morajo biti vključene v BPMN 2.0 in jih označuje z "-", medtem ko lahko storitve označene s "+" direktno uporabimo iz pogonov BPMN. Za podporo integracijskih paketov/adapterjev v pogonih poslovnih procesov BPMN pokažemo, kako se elementi delovnih tokov orkestratorjev oblaka (tj. aktivnosti in nadzorne komponente delovnih tokov) preslikujejo v elemente BPMN 2.0, pri čemer upoštevajo predlagane elemente BPMN, ki so bili oblikovani s pomočjo podprtega razširitvenega mehanizma. Da bi ponudili podporo za upravljanje kompenzacij v orkestratorjih oblaka, smo si zastavili cilj, uporabiti obstoječe elemente BPMN za zapolnitev te vrzeli.

5.5.1 Razširitve BPMN 2.0 s specifikami oblaka

Metamodel BPMN 2.0 ponuja razširitveni mehanizem za obogatitev opisa procesne semantike in njegove diagramske reprezentacije. Za razliko od UML (Unified Modeling Language) Profiles [200, 201], ki uporabljajo pristop razširitve po specifikaciji (angl. Extension by Specification), BPMN podpira pristop razširitve z dodajanjem (angl. Extension by Addition), ki temelji na pripenjanju novih domensko-specifičnih elementov k vnaprej definiranim elementom jezika. Pravzaprav so lahko elementi BPMN razširjeni na dva različna načina. Prvi način je z uporabo modelov BPMN, ki uporabljajo XSD kot izmenljivi format in uporablja elemente *xsd:any* ter *xsd:anyAttbibute*. Drugi način je z uporabo preslikav XMI (XML Metadata Interchange [202]), kjer uporabimo element *xmi:extension*. Ker specifikacija BPMN 2.0 ponuja dvosmerno transformacijo XSLT (EXtensible Stylesheet Language Transformations) med izmenljivimi formati XSD in XMI, je razširitev BPMN 2.0 z uporabo enega izmed le-teh zadostna za implementacijo razširitev v drugem formatu.

Za implementacijo aktivnosti delovnih tokov s specifikami oblaka (kot je kategorizirano v poglavju 5.4.1) uporabimo razširitveni mehanizem BPMN 2.0 in tako dosežemo obogatitev reprezentacije procesnih diagramov. Obstajajo različni tipi opravil, ki so identificirani znotraj BPMN za ločevanje tipov z inherentnim obnašanjem, tj. *Service Task*, *Send Task*, *Receive Task*, *User Task*, *Manual Task*, *Business Rule Task*, *Script Task* ter *Abstract Task*. Seznam opravil je lahko razširjen skupaj z vsemi ustreznimi kazalniki [64]. Iz tega razloga razširimo seznam z dvema novima tipoma opravil: *Cloud-enabled Business Task* (slika 5.12) ter *Cloud-enabled Technical Task* (slika 5.13), da bi vizualizirali poslovne in tehnične zahteve aktivnosti delovnih tokov, specifičnih za oblak. *Cloud-enabled Business Task* je opravilo, ki uporablja nabor vnaprej konfiguriranih sto-

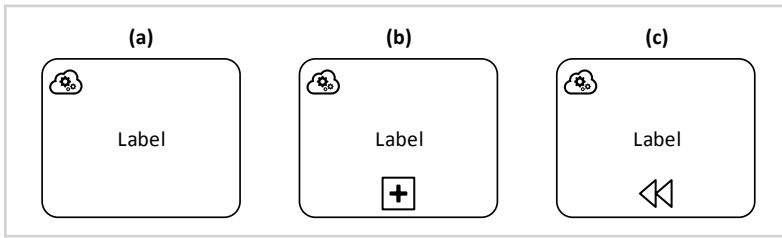


Slika 5.12

Vizualizacija poslovnih aktivnosti delovnih tokov s specifikami oblaka.

ritev oblaka in je lahko spletna storitev (REST/SOAP) ali avtomatizirana skripta na poslovnem nivoju (npr. JavaScript). Te vnaprej konfigurirane storitve oblaka vključujejo: zahteva storitve oblaka, validiranje storitve oblaka, potrjevanje storitve oblaka, analiziranje storitve oblaka, izpolnjevanje storitve oblaka, dokončanje storitve oblaka, upravljanje izpadov storitve oblaka, spremljanje storitve oblaka, obveščanje deležnikov oblaka itn. Objekt *Cloud-enabled Business Task* ima enako obliko kot običajno opravilo, ki je pravokotnik z zaobljenimi robovi, vendar ima grafični marker na zgornjem levem kotu obliko "oblaka s poslovno zgradbo", ki označuje, da je opravilo tipa *Cloud-enabled Business Task* (slika 5.12). *Cloud-enabled Business Task* je pravokotnik z zaobljenimi robovi, ki mora biti narisan z enojno tanko črto in vključuje marker "oblaka s poslovno zgradbo", ki razlikuje ta tip od ostali tipov opravil. Ta marker je lahko uporabljen na atomarni aktivnosti (a), kompozitni aktivnosti (tj. podprocesu) (b), ali aktivnosti s kompenzacijskim markerjem (c).

Cloud-enabled Technical Task je opravilo, ki uporablja nabor vnaprej konfiguriranih storitev oblaka, ki so lahko spletne storitve (REST/SOAP) ali avtomatizirane sistemske skripte (npr. PowerShell [203] ali PowerCLI [204]). Te vnaprej konfigurirane storitve oblaka so lahko kategorizirane v sedem skupin aktivnosti: upravljanje virtualizacije, upravljanje konfiguracij, upravljanje operacij, upravljanje zaščite podatkov, upravljanje virtualnih strojev, upravljanje storitev in upravljanje javnega/hibridnega oblaka. Primer: upravljanje virtualnih strojev vključuje vnaprej konfigurirane tehnične aktivnosti, kot so: upravljanje knjižnic virtualnih strojev, kreiranje virov za virtualne stroje, kot so diski, virtualni trdi diski in omrežni adapterji, izdelava virtualnih strojev iz predlog (angl. Templates), modifikacija obstoječega virtualnega stroja, zagon in zaustavitev virtualnega stroja, ponovni zagon virtualnega stroja, migracija virtualnih strojev na novo vozlišče za upravljanje razpoložljivosti in performance, kreiranje in obnova posnetkov virtualnih strojev itn. Primeri vnaprej konfiguriranih tehničnih aktivnosti v kategoriji



Slika 5.13

Vizualizacija tehničnih aktivnosti delovnih tokov s specifikami oblaka.

upravljanja javnega/hibridnega oblaka so: izdelava nove storitve oblaka, nalaganje paketa v objektno shrambo, postavitev aplikacije iz objektne shrambe, migracija storitve oblaka v javni oblak in obratno itn. Načrtovalec in razvijalec definirata spletno storitev ali skripto v jeziku, ki ga pogon delovnih tokov oblaka zna interpretirati. Ko je opravilo pripravljeno za začetek, pogon delovnih tokov oblaka izvrši spletno storitev/skripto. Ko je spletna storitev/skripta zaključena, je *Cloud-enabled Technical Task* prav tako zaključen. Objekt *Cloud-enabled Technical Task* uporablja isto obliko kot opravilo, ki je pravokotne oblike z zaobljenimi robovi. Poleg tega je v zgornjem levem kotu marker v obliki "oblaka z notranjimi koleščki", ki označuje, da je opravilo tipa *Cloud-enabled Technical Task* (slika 5.13). To opravilo je pravokotnik z zaobljenimi robovi, ki mora biti narisan z enojno tanko črto in mora vključevati marker v obliki "oblaka z notranjimi koleščki", ki diferencira ta tip od ostalih tipov opravil. Ta marker je lahko uporabljen na atomarni aktivnosti (a), kompozitni aktivnosti (tj. podprocesu) (b), ali aktivnosti s kompenzacijskim markerjem (c). Oba tipa novo-predlaganih elementov BPMN (tj. tehnični in poslovni) sta bila razširjena z upoštevanjem dovoljenih razširitvenih pravil, in sicer (1) dovoljena je vključitev markerjev in indikatorjev k obstoječim grafičnim reprezentacijam, (2) dovoljena je zasnova novih oblik elementov, ki niso v konfliktu z obstoječimi in (3) dovoljena je uporaba poljubnih barv in sprememba stila črt, ki ne sme biti v konfliktu z obstoječimi stili črt z namenom, da bi poudarili nov semantični pomen predstavljenega elementa [205].

Za zaključek poudarimo, da razširitveni mehanizem BPMN 2.0 ponuja razvijalcem orodij za procesno modeliranje način za dopolnitev pomanjkljivosti jedrnih elementov BPMN. Z implementiranimi razširitvami lahko procesni model BPMN vizualizira nove aktivnosti delovnih tokov s specifikami oblaka, in tako podpre tehnične in poslovne aktivnosti v oblaku.

5.5.2 Preslikava na BPMN 2.0

Tabela 5.2 prikazuje preslikavo obstoječih aktivnosti in elementov nadzora delovnega toka orkestratorjev oblaka na elemente BPMN 2.0. Začetna točka se preslika ali v (1) *None Start Event*, ki nima definiranega proženja, saj ne obstaja specifičen podrazred *EventDefinition*, ki bi bil asociiran s tem dogodkom; (2) *Message Start Event*, ki ima asociiran *EventDefinition* z začetnim dogodkom. V tem primeru sporočilo prispe iz participanta in proži začetek procesa; ali (3) *Timer Start Event*, ki omogoča nastavljanje specifičnega začetnega časa-datuma ali specifičnega cikla (npr. vsak dan ob 10:00), ki proži začetek delovnega toka. V tem primeru je asociiran zgolj en *EventDefinition* z začetnim dogodkom in ta *EventDefinition* je podrazred *TimerEventDefinition*. Končna točka se preslika ali v (1) *None End Event*, ki nima definiranih prožilcev, saj ne obstaja specifičen podrazred *EventDefinition*, ki je asociiran s tem dogodkom; ali (2) *Message End Event*, ki ima asociiran *EventDefinition* z začetnim dogodkom, v tem primeru prispe sporočilo od participanta in proži začetek procesa. Povezava se preslika ali v (1) *Sequence Flow*, kjer je privzeta vrednost *false* in *conditionExpression* ni specificiran; (2) *Conditional Sequence Flow*, kjer je privzeta vrednost *false* in je specificiran *conditionExpression*, ali (3)*Default Sequence Flow*, kjer je privzeta vrednost *true* in *conditionExpression* ni specificiran. Zanke se preslikajo v *Loop Activity*, ki služijo kot ovojnica (angl. Wrapper) za notranjo aktivnost (atomarno ali gnezdeno), ki je lahko izvedena večkrat zaporedno.

Standardna aktivnost se preslika v *Script Task*, ki je zadolžen za klicanje asociiranih skript ob aktivaciji (običajno gre za API-je storitvenih modelov IaaS in PaaS). Ko se skripta enkrat uspešno izvede, se *Script Task* zaključi. Aktivnost za spremljanje se preslika v *Receive Task*, ki je oblikovana za čakanje na prihod sporočila iz zunanjih participantov (npr. drugega oblaka). Ko je sporočilo enkrat prejeto, je opravilo zaključeno in delovni tok se lahko nadaljuje. Aktivnosti, specifične za oblak, se preslikajo ali v *Cloud-enabled Business Task* ali v *Cloud-enabled Technical Task*, pri čemer sta obe opravili razširjeni iz standardnega seznama tipov opravil z razlogom, da bi vizualizirali poslovne in tehnične zahteve aktivnosti delovnih tokov s specifikami oblaka. Kostumizirane aktivnosti se preslikajo v standardna skriptna opravila, ki so razširjena in prilagojena vsaki skupini uporabnikov s ciljem, da bi zadostili njihovim specifičnim zahtevam (npr. zahtevamo funkcionalnost, ki ni na voljo znotraj platforme orkestracije). Gnezdena aktivnost se preslika v podproces (angl. Sub-Process), katerega začetne

Tabela 5.2

Preslikava obstoječih elementov delovnih tokov na elemente BPMN 2.0.

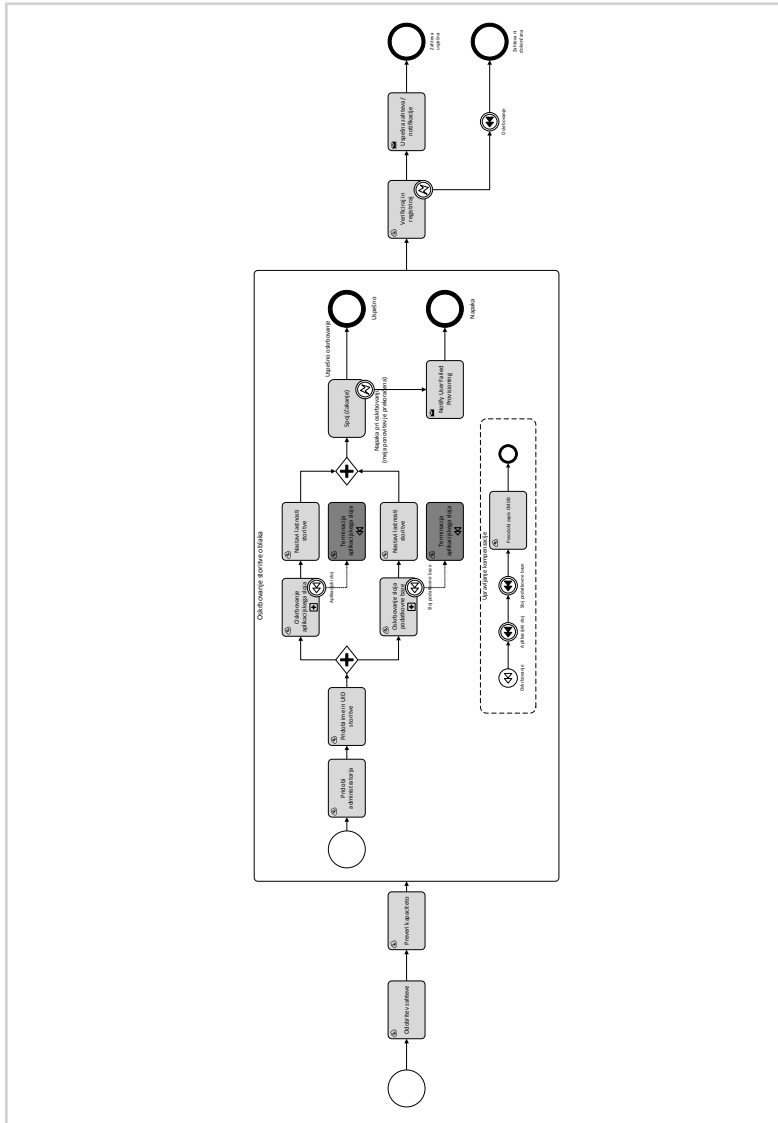
Delovni tokovi orkestratorjev oblaka	***Elementi BPMN 2.0***	***Vizualizacija***
Nadzor delovnega toka		
Začetna točka	None Start Event	
	Message Start Event	
	Timer Start Event	
Končna točka	None End Event	
	Message End Event	
Povezave	Sequence Flow	
	Conditional Sequence Flow	
	Default Sequence Flow	
Zanke	Loop Activity	
Aktivnosti		
Standardne	Script Task	
Aktivnosti za spremljanje	Receive Task	
Specifične za oblak	**Cloud-enabled Business Task**	
	Cloud-enabled Technical Task	
Kostumizirane	Extended standard Script Tasks	
Gnezdene	Sub-process	

podrobnosti so bile modelirane z uporabo aktivnosti, prehodov, dogodkov in zaporednih tokov. Gnezdena aktivnost definira kontekstualni doseg, ki ga lahko uporabimo za definiranje vidljivosti atributov (angl. Attribute Visibility), doseg transakcij, za obravnavo izjem in dogodkov ter za kompenzacije.

V naslednjem primeru (slika 5.14) prikažemo, kako lahko vpeljemo tehnike upravljanja kompenzacij iz primera na sliki 5.5 z uporabo notacije BPMN 2.0. Ko smo enkrat prejeli potrebne odobritve in validirali zadostne kapacitete za procesiranje zahteve, se morajo izvesti aktivnosti povezane z oskrbovanjem storitve oblaka. V prvem koraku tehnična aktivnost pridobi ustrezne poverilnice računa administratorja in UID storitve ter ime, ki so potrebni za nadaljnjo postavitev storitve. Za tem se morata izvesti dve vzporedni aktivnosti: (1) oskrbovanje aplikacijskega sloja in (2) oskrbovanje sloja podatkovne baze. Zaradi lažjega razumevanja in enostavnosti v našem primeru nismo upoštevali vrstnega reda postavitve posameznih storitvenih slojev (npr. sloj podatkovne baze mora biti postavljen prvi, saj ostali sloji zahtevajo povezljivost s podatkovno bazo preko naslova IP). Ko so lastnosti storitve enkrat nastavljene za oba sloja, se sloji obravnavajo kot postavljeni. Sprejeti so ukrepi, ki v primeru napak pri aktivnostih oskrbovanja storitve zagotavljajo razveljavitve postavitev. V primeru, da pride do napake med aktivnostjo spoja, se postopek oskrbovanja ponavlja, dokler ni presežena meja dovoljenih ponovitev. Ko sta oba sloja uspešno postavljena, se preiskovanje delovnega toka nadaljuje. Uspešnemu oskrbovanju sledi verifikacija/registracija zahtevane storitve, pri čemer se delovni tok zaustavi in so poslane notifikacije vsem deležnikom o uspešnem zaključku storitvene zahteve. Če pride do napake med aktivnostjo "verificiraj in registriraj" (na primer registracija novo ustvarjenega sredstva v CMDB je bila neuspešna zaradi napake pri povezavi do podatkovne baze), sta postavitvi aplikacijskega sloja in sloja podatkovne baze povrnjeni/razveljavljeni, pri čemer je zapis v CMDB ustrezno posodobljen.

5.5.3 *Proof-of-concept*

Naš generični pristop upravljanja kompenzacije, ki je aplikabilen nad orkestratorji oblaka skupaj z algoritmom CAS, je bil verificiran z implementacijo PoC. Implementiramo sistem, ki je zmožen opravljati dvosmerno razčlenjevanje (angl. Bidirectional Parsing), tj. iz reprezentacije delovnih tokov v obliki dinamičnih objektov Java v format izmenjave BPMN 2.0 (XML) ter obratno (iz formata izmenjave BPMN 2.0 v reprezentacijo delovnih tokov v obliki dinamičnih objektov Java). Ta operacija je izvedena



Slika 5.14

Primer uporabe adaptiranja BPMN 2.0 za orkestracijo opravil, specifičnih za oblak.

z razčlenjevalnikom delovnih tokov (angl. Workflow Parser), ki uporablja tehnologijo Java Architecture for XML Binding (JAXB). Treba je vedeti, da je reprezentacija delovnih tokov v obliki objektov Java obogatena s konstrukti upravljanja kompenzacij in uporablja strukturo seznam sosednosti za shranjevanje in dinamično konstruiranje delovnih tokov v obliki prilagojenega usmerjenega grafa.

Definicija 1: Seznam sosednosti je reprezentacija usmerjenega grafa z n vozlišči, ki uporablja polje n seznamov vozlišč. Seznam i vsebuje vozlišče j, če obstaja povezava iz vozlišča i do vozlišča j.

Seznam sosednosti je priporočljiva reprezentacija grafov, ko so le-ti skopi/redki in zatorej predstavljajo primeren način za reprezentacijo/shranjevanje delovnih tokov, specifičnih za oblak. Ti so sestavljeni iz polja kazalcev do naslednjih vozlišč, vsak kazalec za svoje vozlišče v grafu. Poglejmo si primer usmerjenega grafa s štirimi vozlišči (slika 5.3). Puščica (→) predstavlja povezavo znotraj seznama, pri čemer (NULL) pomeni, da vozlišče nima nobenega naslednika. Na tej točki je lahko seznam sosednosti prikazan na naslednji način:

$$1 \rightarrow 2 \rightarrow 3$$
$$2 \rightarrow 4$$
$$3 \rightarrow 4$$
$$4 \rightarrow \text{NULL}$$

Z upoštevanjem seznama sosednosti in algoritma CAS, razširimo standardna digraf vozlišča s spremenljivkami, ki so prikazane potemnjeno v tabeli 5.3. Seznam sosednosti predhodnih vozlišč (*List<Node> predecessorNodes*) je potreben, saj naš predlagan algoritem izvaja reverzne obhode, pri čemer začne preiskovanje pri končnih vozliščih. Spremenljivka *hasAttachedCompensationHandler* označuje prisotnost pripetega (mejnega) kompenzacijskega dogodka *catch*, medtem ko sta spremenljivki *isNested* in *nestedWorkflow* koristni v primeru vgrajenih aktivnosti.

Implementiramo razčlenjevalnik delovnih tokov in algoritem CAS (kot je predlagan v poglavju 5.4.2) z uporabo programskega jezika Java in ogrodja Eclipse Integrated Development environment (IDE)(slika 5.15). Algoritem je bil razvit kot del paketa

Tabela 5.3

Razširitev standarnih vozlišč digrafa.

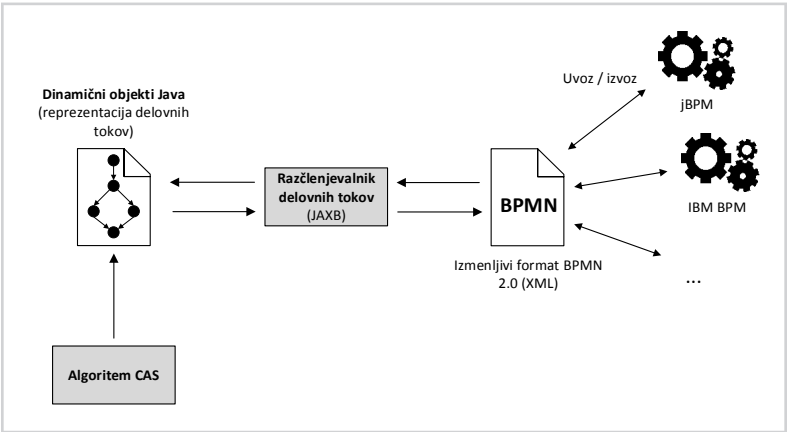
Spremenljivke	***Opis***
String data	Podatkovni element, ki hrani skripto, specifično za oblak.
boolean visited	Zastavica, ki označuje že obiskana vozlišča.
List<Node> adjacentNodes	Seznam sosednosti, ki hrani vsa naslednja vozlišča.
List<Node> predecessorNodes	Seznam sosednosti, ki hrani vsa predhodna vozlišča.
boolean hasAttachedCompensationHandler	Zastavica označuje ali je vozlišču pripet upravljavec kompenzacij.
boolean isNested	Zastavica označuje gnezdeno vozlišče znotraj delovnega toka.
Graph nestedWorkflow	Objekt tipa *Graph*, ki predstavlja gnezden delovni tok.

si.cloud.cas, ki ga sestavljajo trije med seboj odvisni javanski razredi: *CASAlgorithm*, *Graph* in *Node*. *CASAlgorithm* vsebuje metodo *compensationActivitySearch(Graph workflow):List<Node>*, ki izvaja dejanski algoritem CAS, in metodo *returnList(List<Node> list):void*, ki prikazuje seznam aktivnosti za kompenzacijo. Razred *Graph* hrani dva seznama, seznam vseh vozlišč v delovnem toku in seznam vseh končnih vozlišč, medtem ko razred *Node* hrani prej omenjene spremenljivke in objekte (iz tabele 5.3). Efektivnost našega pristopa pokažemo tako, da testiramo algoritem CAS na številnih poljubno generiranih delovnih tokovih, z naključno pripetimi upravljavci kompenzacij (tako v atomarnih kot tudi gnezdenih vozliščih). Naš algoritem je uspešno detektiral vse upravljavce kompenzacij, ki segajo do poljubne globine in velikosti delovnih tokov. Razčlenjevalnik delovnih tokov je predstavljen kot ločen paket (*si.cloud.parser*). Glavni javanski razred iz tega paketa (*WorkflowParser*) vsebuje dve ključni metodi:

1. *convertToGraph(File bpmnFile):Graph* – razčlenjevalnik delovnega toka uporablja JAXB za generiranje reprezentacije delovnih tokov v obliki dinamičnih objektov Java (tj. modificirani digrafi) iz XML dokumenta (BPMN 2.0 Interchange Format),

2. *convertToBPMN(Graph workflow):File* – razčlenjevalnik delovnih tokov uporablja JAXB za generiranje dokumenta XML (BPMN 2.0 Interchange Format) iz reprezentacije delovnih tokov v obliki dinamičnih objektov Java.

S prvo operacijo pokažemo, da je naš generični delovni tok lahko transformiran v razširjen BPMN 2.0 izmenljivi format in je potemtakem lahko uvožen in izvajan na



Slika 5.15

Arhitekturni diagram sistema PoC.

poljubnem procesnem strežniku, ki je skladen z BPMN 2.0 (npr. jBPM [206] in IBM BPM [207]), vključno z razširjenimi aktivnostmi delovnih tokov s specifikami oblaka. Z drugo operacijo pokažemo, da so razširjeni procesi BPMN 2.0 lahko transformirani v generične delovne tokove, ki so nato lahko uporabljeni za (1) avtomatizacijo opravil, specifičnih za oblak in (2) izvajanje algoritma CAS ter klicanje kompenzacijskih aktivnosti. Skratka, sistem PoC skupaj s primerom uporabe pokaže efektivnost in izvedljivost predlaganega pristopa, ki evidentno povečuje učinkovitost pri oblikovanju delovnih tokov, ki so odpornejši na napake.

"A journey of a thousand miles begins with a single step."
— Lao Tzu

6

Sorodne raziskave in diskusija

Skladno s trinivojsko strukturo referenčnega modela integracije infrastrukturnih in platformskih nivojev računalniškega oblaka klasificiramo sorodne raziskave glede na tri komplementarne modele: arhitekturni model IaaS, model obogatitve sodobnih izvajalnih platform s specifikami oblaka ter model upravljanja kompenzacij v orkestracijskih platformah oblaka.

6.1 Arhitekturni model IaaS

Večina današnjih rešitev IaaS je komercialnih, vendar kljub temu obstajajo številni odprto-kodni produkti tako v industriji kot tudi v akademskem svetu. Zaradi povečanega interesa po adapciji takšnih ogrodij v okoljih IT, so mehanizmi za zagotavljanje poštene primerjave in odločanja dobili nov zagon. Taksonomija in ogrodje enotne in celovite arhitekture še nista bila naslovljena na način, ki bi bil primerljiv pristopu predlaganemu v tej disertaciji. Kljub temu so arhitekture IaaS že bile analizirane in primerjane v številnih študijah [21, 29–38]. B. Sotomayor et al. [21] v svojem raziskovalnem delu izvedejo primerjavo orodij, ki ponujajo zmožnosti upravljanja virtualne infrastrukture, pri čemer so podrobno primerjali platformo OpenNebula s številnimi dobro znanimi upravitelji virtualne infrastrukture, kot sta Eucalyptus in Nimbus. Avtorji so za primerjavo uporabili samo pet osnovnih parametrov (npr. podpora za hibridni oblak in oddaljene vmesnike), pri čemer niso upoštevali ostale vitalne sloje arhitekture IaaS, kot so predlagani v naši raziskavi. Ravno tako niso predlagali taksonomije ter arhitekturnega ogrodja.

M. Mahjoub et al. [29] so naredili raziskavo trenutnih ponudnikov oblaka in tehnologij z namenom podpore izbiri najboljšega ponudnika oblaka, ki izpolnjuje vse uporabnikove potrebe za izgradnjo lastne infrastrukture oblaka z najbolj primerno odprtokodno tehnologijo IaaS. Primerjali in analizirali so vse večje igralce iz sveta odprtokodnih rešitev IaaS, pri čemer so upoštevali devet kriterijev. Kljub temu njihova primerjava ni upoštevala nekaterih bistvenih komponent ogrodij IaaS, kot so repozitorij slik, storitev identitete ter nadzorna plošča. Naša analiza je bila narejena z uporabo 44 parametrov, ki temeljijo na taksonomiji. Naslednja primerjava in analiza odprtokodnih sistemov IaaS je bila izvedena s strani avtorjev P. Sempolinski in D. Thain [30], ki primerjata tri rešitve (Eucalyptus, OpenNebula in Nimbus) ter identificirata štiri primerjalne kriterije: zmožnosti diskovnih slik, shramba diskovnih slik, hipervizorji in unikatne funkcionalnosti. Mi predlagamo celovit nabor kriterijev, ki temeljijo na taksonomiji in celostni arhitekturi IaaS. Karakteristike, arhitekture in aplikacije

številnih popularnih platform računalništva v oblaku so v svojem raziskovalnem delu analizirali in diskutirali J. Peng et al. [31]. Čeprav so arhitekture odprto-kodnih rešitev IaaS podrobno artikulirane, primerjava ni naslovila pomembnih aspektov, kot so varnost ter upravljanje z oblakom. I. Voras et al. [32] so izpeljali nabor kriterijev za evalvacijo in primerjavo rešitev IaaS, ki so lahko uporabljeni za ocenjevanje in izbor ustreznih produktov. Čeprav so kriteriji organizirani v šest kategorij, ne primerjajo nekaterih kritičnih podpornih aspektov, kot je repozitorij slik, storitev prenosa, vmesniške komponente (tj. orodja CLI, API-ji, orkestrator in nadzorna plošča) in komponente upravljanja SLA-jev. Ti aspekti so bili naslovljeni skupaj s predlagano taksonomijo in arhitekturnim ogrodjem v našem delu. S. Wind [33] je izpeljal kriterije iz literarne analize in primerja obstoječe odprto-kodne upravljavske platforme računalništva v oblaku. Pomembni visokonivojski kriteriji so upoštevani (tj. varnost, interoperabilnost, uporabniški vmesnik itn.), vendar primerjava ne vključuje nizkonivojskih kriterijev, kot so viri (npr. virtualno omrežje in nosilci) in podpora upravljanju, kot je to naslovljeno v našem delu. Naslednjo raziskavo, ki komparativno opisuje popularne rešitve oblaka, so predstavili T. Cordeiro et al. [34]. Čeprav njihovo delo diskutira razlike med posameznim produktom in podaja ilustrativne primere uporabe, ni bil izpeljan noben kriterij, glede na katerega bi bili produkti primerno primerjani in poravnani. M. Rodriguez-Martinez et al. [35] poročajo o evalvaciji odprto-kodnih orodij za računalništvo v oblaku, ki temeljijo na matriki zmožnosti (angl. Capability Matrix) uporabljeni za karakterizacijo številnih funkcionalnosti. Matrika vsebuje deset različnih kriterijev, kot so upravljavska orodja, razvojna orodja, skalabilnost, razpoložljivost itn. Kljub temu izpeljani kriteriji ne implicirajo poštene primerjave, saj je zgolj en izmed primerjanih produktov, dejanski odprto-kodni produkt IaaS (tj. Eucalyptus). Nasprotno, naša primerjava temelji na 44 kriterijih, ki so strukturirani znotraj taksonomije in podprti s strani arhitekturnega ogrodja. D. Ogrizovic et al. [36] predstavijo nekaj znanstvenih oblakov, ki so zgrajeni z uporabo ogrodij distribuiranega računalništva, ne da bi pri tem izvedli kakršnokoli analizo ali primerjavo teh rešitev, kot to počnemo v našem delu. L. Ang et al. [38] so z vidika performance izvedli celovito primerjavo javnih ponudnikov oblaka ter sistematično določili, katere metrike najboljše karakterizirajo zmogljivost oblaka. Kljub temu, evalvacijske metrike naslavljajo najbolj pomembne aspekte performance, medtem ko ne predstavljajo splošno-namenski primerjalni sistem, kot je to storjeno v našem delu.

Taksonomije računalniških oblakov so že bile definirane v številnih delih [12, 208–212]. National Institute of Standards and Technology [12], B. P. Rimal et al. [208] in Intel [212] predstavljajo splošno taksonomijo računalništva v oblaku. V nasprotju z našo taksonomijo, oni ne naslavljajo vitalnih zmožnosti IaaS, kot so storitve z dodano vrednostjo. Na drugi strani A. Beloglazov et al. [209] vpeljujejo taksonomijo energijsko-učinkovite zasnove računalniških oblakov. Naše delo se ne osredotoča na energijsko učinkovitost, vendar naslavlja taksonomijo na nivoju infrastrukture in pokriva vse bistvene karakteristike platform IaaS. Forrester [210] vpeljuje tržno-usmerjeno taksonomijo računalništva v oblaku in ne vpeljuje tehničnih zmožnosti platform IaaS, kot je to diskutirano v naši raziskavi. OpenCrowd [211] predstavlja splošno taksonomijo računalništva v oblaku. Taksonomija prav tako naslavlja sloj IaaS in identificira štiri komponente: shrambo, računsko moč, upravljavske storitve in posrednika oblaka (angl. Cloud Broker). V nasprotju s tem naša taksonomija naslavlja 44 komponent in jih kategorizira v urejene sloje. Pregled sorodnih raziskav je pokazal, da sta predlagana taksonomija in arhitekturno ogrodje IaaS, skupaj z evalvacijo najpomembnejših platform IaaS, nova prispevka v znanosti, ki še nista bila obravnavana na podoben način.

6.2 *Model obogatitve sodobnih izvajalnih platform s specifikami oblaka*

Razširitve sodobnih izvajalnih platform, ki so specifične za oblak, še niso bile naslovljene na način, ki bi bil primerljiv pristopu, predlaganemu v naši disertaciji. Kljub temu so bile narejene številne raziskave, ki obravnavajo aspekt elastičnosti in nadzornih sistemov oblaka. Kot rezultat klasificiramo sorodne raziskave modela integracije IaaS in PaaS v:

1. programske direktive in politike, ki so specifične za oblak [213, 214],
2. aspekte elastičnosti sistemov IaaS in PaaS [22, 28, 48, 55, 135, 137, 152, 215],
3. nadzorniške funkcije oblaka [171, 216–222].

Vaquero LM et al. [213] opisujejo arhitekturo za dinamično nadziranje obnašanja postavljenih aplikacij v oblaku z uporabo visokonivojskih pravil. Njihova arhitektura temelji na kostumiziranem pogonu pravil (angl. Rule Engine), ki omogoča izvajanje pravil za diktiranje obnašanja aplikacij za oblak in omogoča ponudnikom aplikacij,

da ponovno definirajo (angl. Redefine) vedenjske politike v času izvajanja. Pri tem se lahko uporabniki prilagajajo na nove pogoje/zahteve poslovanja in spreminjajoče se obremenitve. Kljub temu arhitektura razširja izvorni predlog formata OVF ter formata RIF (Interchange Format) in na tak način povečuje kompleksnost razumevanja iz perspektive razvijalca in aplikacijskega administratorja. V nasprotju s tem mi propagiramo aspekte elastičnosti na nivo aplikacije in platforme in na tak način ponujamo administratorjem aplikacije in razvijalcem večji nadzor in boljše upravljanje, pri čemer skrijemo kompleksnost spodaj ležečih infrastrukturnih sistemov. Na podoben način S. Dustdar et al. [214] aplicirajo princip programskih direktiv za upravljanje notranje elastičnosti računalništva v oblaku, medtem ko omogočajo nadzor virov, stroškov ter kakovosti. Še več, njihov pristop omogoča razvijalcem vključevanje programskih direktiv v sodobne programske jezike, s katerimi nadzirajo elastičnost aplikacije. Čeprav za doseganje elastičnosti naslavljajo omejevanje virov, stroškov in kakovosti, ne obravnavajo metrik, ki so specifične za aplikacijo in omogočajo pridobivanje relevantnih performančnih informacij iz določenih slojev aplikacije (npr. število aktivnih sej na spletni strežnik). Cilj njihovih direktiv SYBL je ponuditi abstraktni nabor programskih direktiv, ki so lahko kombinirane z večino splošno-namenskimi jeziki. Naš cilj je definirati generičen sistem parametrov, ki ga lahko uporabimo za oblikovanje obogatitve sodobnih izvajalnih platform, ki temeljijo na oblaku, tako da uporabljamo politike in programske direktive in kot rezultat naslavljamo deficitne/pomanjkljive zmožnosti elastičnosti in nadzora oblaka.

Večina trenutnih ponudnikov PaaS, kot so Amazon Elastic Beanstalk [160], Google App Engine [158], Aneka [223] in Windows Azure [159], ne ponuja podpore za postavitev in konfiguracijo storitve oblaka na nivoju aplikacije in platforme (tako samodejno kot tudi ročno). Prav tako ne podpirajo zmožnosti nadzora oblaka, ki so izpostavljene v poglavju 4.3.1, in bi jih lahko upravljali in nadzirali preko programskih direktiv in politik ter s tem omogočili razvijalcem, da se osredotočijo na definicijo primernih deskriptorjev znotraj njihovih aplikacij, ki se izvajajo v oblaku. Dodatno, J. Kirschnick et al. [55] opisujejo arhitekturo, ki omogoča avtomatizirano postavitev in upravljanje virtualne infrastrukture ter programske opreme postavljenih storitev v oblaku. Njihova arhitektura vzame opis predloge storitve oblaka, ki ograjuje zahteve, možnosti, kot tudi obnašanje kolekcije virov, in orkestrira oskrbovanje teh virov v novo ustvarjen nabor virtualnih virov. Čeprav avtorji uporabljajo nabor domensko-specifičnih jezikov (angl. Domain-Specific Languages – DSLs) za opis želenega stanja virtualne infrastrukture in

komponent programske opreme za storitve oblaka, ne inkorporirajo funkcij elastičnosti in nadzora oblaka na nivo aplikacij in platforme, kot je to predstavljeno v našem delu. Calheiros RN et al. [215] predstavijo arhitekturo, načrt in evalvacijo koordinatorja oblaka (angl. Cloud Coordinator) iz arhitekture InterCloud, ki omogoča brezhibno izvrševanje SLA-jev elastične aplikacije, tako da jih skalira med različne podatkovne centre. Kljub temu se avtorji osredotočajo izključno na infrastrukturno skalabilnost in ne naslavlja aspektov elastičnosti na nivo aplikacije in platforme. Marshall P et al. [152] predlagajo model elastičnega sistema, ki se je zmožen odzivati na spremembe zahtev, tako da uporablja nove vire oblaka, vendar so v svojih raziskavah pokrili samo elastičnost na nivoju infrastrukture. Mi predlagamo generičen model elastičnosti, ki naslavlja elastičnost storitve oblaka, pri čemer upoštevamo aplikacijo kot celoto. Vaquero LM et al. [48] so predstavili iniciativo za skalabilnost celotne aplikacije v okolju oblaka, kjer so naslovili skalabilnost na nivoju IaaS, kot tudi na nivoju PaaS. Kljub temu avtorji ponujajo zgolj celoviti pregled trendov elastičnosti na različnih nivojih, medtem ko mi na inovativen način vpeljujemo zmožnosti elastičnosti na nivo aplikacij in platforme. Nov sloj abstrakcije, ki je bližje življenjskemu ciklu storitev in omogoča samodejno postavitev in eskalacijo glede na status storitve (ne samo na infrastrukturi) je bil predlagan s strani avtorjev Rodero-Merino et al. [22]. Le-ti vpeljujejo mehanizem samodejne elastičnosti na nižjem nivoju, tako da razširjajo specifikacijo OVF. Nasprotno pa mi propagiramo funkcionalnosti elastičnosti na nivo aplikacije in platforme, medtem ko vpeljujemo nove programske direktive/politike in razširjamo obstoječe. Avtorji članka [135] predlagajo razslojen nabor API-jev, ki ponujajo suplementarno stopnjo svobode, od programskih jezikov do stilov. Razvili so implementacijo PoC v Javi in ga testirali kot del projekta mOSAIC, vendar se API osredotoča na reševanje težav portabilnosti oblakov PaaS in ne obravnava funkcionalnosti elastičnosti storitev oblaka. Podoben evropski projekt RESERVOIR [28] eksplicitno naslavlja omejeno skalabilnost oblaka enojnega ponudnika (angl. Single-Provider Cloud), omejeno interoperabilnost med ponudniki oblaka, ter pomanjkanje vgrajene podpore za upravljanje poslovnih storitev (angl. Business Service Management) pri trenutnih ponudnikih oblaka. Čeprav so naslovljene funkcionalnosti elastičnosti, je upoštevan zgolj nivo IaaS. V nasprotju s slednjim pa mi naslavljamo celovit sistem elastičnosti in ga vpeljemo na nivo aplikacije in platforme, medtem ko predlagamo razširitve platformskih vsebnikov s specifikami oblaka. Nazadnje, avtorji [137] predlagajo pristop elastičnega skaliranja, ki uporablja stroškovne kriterije za detektiranje in analizo ozkih grl znotraj večslojnih aplikacij za

oblak. Kljub temu v svoje sisteme elastičnosti ne inkorporirajo aspektov kvalitete, tj. uporabniške izkušnje, kot je to elaborirano v našem delu.

B. Konig et al. [171] so predstavili skalabilni in elastični, distribuiran sistem za spremljanje infrastrukture oblaka, ki temelji na arhitekturi vsak z vsakim (angl. Peer-To-Peer – P2P). Ta arhitektura omogoča postavitev dolgo živečih poizvedb v omrežju z namenom spremljanja nabora metrik, ki se raztezajo čez vse sloje sklada oblaka, in omogočajo agregiranje nizkonivojskih metrik iz operacijskih sistemov v visokonivojske, aplikacijsko-specifične metrike, ki so izpeljane iz storitev, podatkovnih baz ali aplikacijskih logov. Kljub temu pridobivanje poslovnih metrik, kot so stroškovne metrike in metrike kakovosti, niso bile upoštevane in je zaradi tega nemogoče zadovoljiti uporabniške zahteve po performanci in stroških. Mi upoštevamo tehnične in poslovne metrike ter jih inkorporiramo v visokonivojske programske direktive. S.A. De Chaves et al. [216] diskutirajo arhitekturo in implementacijo sistema za spremljanje privatnega oblaka (angl. Private Cloud Monitoring System – PCMONS) ter njihovih aplikacij preko primera uporabe predlagane arhitekture. Načrtovali so abstraktno, splošno-namensko arhitekturo spremljanja, ki je sestavljena iz treh slojev: infrastrukturnega, integracijskega in predstavitvenega sloja. Njihova arhitektura upošteva zgolj nizkonivojske, infrastrukturne metrike. Medtem ko naš predlog opredeljuje tudi višjenivojske, aplikacijsko-specifične metrike, vključno s poslovnimi metrikami, kot so metrike kakovosti in stroškov. J. Spring [217] predstavlja nabor priporočljivih restrikcij in revizij, da bi uveljavil spremljanje varnosti v oblaku, vendar ne opredeljuje metrik, relevantnih za storitev oblaka, uporabniške izkušnje itn., kot je to predstavljeno v našem doprinosu. P. Hasselmeyer in N. d'Heureuse [218] vpeljujeta svoj pristop celovitega sistema za spremljanje podatkovnih centrov svojih strank. Čeprav ponujajo generično propagacijo informacij za spremljanje in shrambo, ki pokriva različne tipe virov na vseh nivojih (omrežju, strežnikih in aplikacijah), vključno z večnajemništvom, ne vpeljujejo relevantnih poslovnih metrik, kot je to storjeno v našem prispevku. M. Anand [219] predstavlja skalabilno ogrodje za spremljanje aplikacij oblaka, tj. Cloud Monitor. Kljub temu to ogrodje spremlja zgolj gručna vozlišča in tako pridobiva nizkonivojske infrastrukturne metrike. V nasprotju z njihovim pristopom, mi predlagamo celosten nabor metrik spremljanja, tako nizkonivojskih kot tudi visokonivojskih, ki predstavljajo še posebno vrednost za razvijalce aplikacije in aplikacijske administratorje. M. Dhingra et al. [220] predlagajo distribuirano ogrodje spremljanja, ki omogoča drobnozrnato (angl. Fine Grained) spremljanje aplikacij in demonstrirajo prototipno

sistemsko implementacijo za tipične primere uporabe. V nasprotju z našim delom, njihov pristop daje velik poudarek nizkonivojskim aspektom spremljanja, pri čemer pridobivajo zgolj infrastrukturne metrike, kot so uporaba CPU ter vhodne/izhodne pasovne širine omrežja. D. Zou et al. [221] so oblikovali zaupanja vredno ogrodje za spremljanje (angl. Trusted Monitoring Framework), ki ponuja verigo zaupanja za izključevanje nezaupljivih privilegiranih domen. To so storili tako, da so postavili neodvisno domeno gosta za namene spremljanja in utilizirali zaupljivo računalniško tehnologijo za zagotovitev integritete okolja za spremljanje. Kljub temu je njihov fokus na zaupanja vrednem spremljanju, medtem ko se mi osredotočamo na agregiranje infrastrukturnih, aplikacijsko-specifičnih, uporabniških in stroškovnih metrik, da bi dosegli večjo stopnjo upravljanja in nadzora aplikacij za oblak. Na podoben način O. Adinolfi et al. [222] diskutirajo zasnovo in implementacijo sistema QoS-MONaaS za spremljanje QoS na nivoju poslovnih procesov nad generično platformo oblaka. Čeprav njihov pristop spremlja metrike kakovosti, jih ne propagirajo na nivo aplikacije in infrastrukture, niti ne dajejo poudarka na poslovne in infrastrukturne metrike. Nasprotno, mi vpeljujemo kombiniran pristop, ki upošteva stroške, kakovost in delovne obremenitve pri procesu samodejnega skaliranja ter vključujemo te aspekte na nivo aplikacije in platforme.

Pregled sorodnih raziskav je pokazal, da je generičen sistem parametrov, skupaj z razširitvami Java EE Annotation Metadata, nov prispevek v znanosti in še ni bil naslovljen na podoben način kot smo ga mi v tej disertaciji.

6.3 Upravljanje kompenzacij v orkestracijskih platformah oblaka

Dogodkovno usmerjena arhitektura orkestratorja, ki je odporna na napake in podpira principe upravljanja kompenzacij, še ni bila naslovljena na način, ki bi bil primerljiv pristopu predlaganem v tej disertaciji. Kljub temu so bile narejene že številne raziskave, ki obravnavajo vpeljavo mehanizmov za zagotavljanje odpornosti/tolerance na napake. Kot rezultat klasificiramo sorodne raziskave v:

1. orkestracijo virov oblaka [54, 56, 59–63, 224–230],
2. znanstvene delovne tokove in orkestracijo [231–233],
3. upravljanje kompenzacij [64, 65].

Orkestracija virov v oblaku opisuje avtomatizirano ureditev, koordinacijo in upravljanje kompleksnih sistemov in virov oblaka. Le-ta je uveljavljena s pomočjo orkestracijskih orodij (tj. orkestratorjev), ki omogočajo načrtovanje poslovnih in tehničnih delovnih tokov. Termin delovni tok označuje avtomatizacijo procedur, kjer so dokumenti, informacije ali opravila posredovana med participanti, skladno z definiranim naborom pravil, da bi dosegli oz. prispevali k skupnemu poslovnemu cilju [197]. Orkestracija oblaka se razlikuje od tipičnega avtomatiziranega procesa delovnih tokov, saj povezuje različne oz. dislocirane avtomatizirane procese in vire oblaka, ki uporabljajo delovne tokove preko portala, s pomočjo katerega so ti delovni tokovi lahko upravljani [224]. Obstajata dva pomembna arhitekturna pristopa za implementacijo delovnih tokov: storitvena orkestracija in koreografija [57]. Orkestracija storitev temelji na centraliziranem pogonu, ki nadzira celoten proces, vključno z nadzornim tokom kakor tudi podatkovnim tokom (npr. BPEL). Na drugi strani koreografija storitev predstavlja obliko storitvene kompozicije, pri kateri je definiran interakcijski protokol med številnimi partnerskimi storitvami iz globalne perspektive, da bi dosegli skupni cilj (npr. Web Services Choreography Description Language – WS-CDL). V tem primeru ni uporabljen centraliziran pogon, saj storitve sodelujejo na kolaborativen način. Kljub temu večina orkestratorjev oblaka uporablja pristop storitvene orkestracije [56, 59–63], saj je koreografija storitev kompleksni proces in manj primerna za adaptacije distribuiranih sistemov oblaka.

V akademskem svetu in industriji je bilo v zadnjih letih veliko pozornosti namenjene izgradnji platform za orkestracijo oblaka. Skoraj vsa večja podjetja IT, npr. Microsoft [56], VMware [59], BMC [60], Cisco [61], HP [62] in IBM [63] so razvili lastniške rešitve orkestratorjev oblaka, ki omogočajo avtomatizacijo storitev oblaka, od začetka do konca. Kljub temu so ti pristopi specifični ponudnikovi tehnologiji in ne vpeljujejo semantike upravljanja kompenzacij v njihove platforme delovnih tokov. Na podoben način Amazon ponuja rešitev Simple Workflow Service (SWF) [230], ki predstavlja upravljavsko storitev delovnih tokov za izgradnjo aplikacij za oblak in ponuja preproste API klice. Ti klici so lahko izvedeni iz kode poljubnih programskih jezikov in se lahko izvajajo na instancah EC2. SWF na tak način deluje kot koordinacijsko središče, s katerim komunicirajo gostitelji aplikacij. Na drugi strani so se na tem področju pojavile številne akademske raziskave. Y. Mao et al. [225] predlagajo podatkovno-usmerjen pristop orkestracije oblaka, kjer so viri oblaka modelirani kot strukturirani podatki, ki jih lahko preiskujemo z deklarativnim jezikom, in jih posodabljamo z do-

bro definirano transakcijsko semantiko. Njihov pristop ne vpeljuje arhitekture upravljanja kompenzacij ter ne naslavlja delovnih tokov za orkestracijo poljubnih storitev oblaka. Na podoben način C. Liu et al. [54] naslavljajo izzive, s katerimi se soočajo med orkestracijo kompleksnega nabora podsistemov (viri računske moči, omrežni viri, viri shrambe), ki se raztezajo med geografskimi regijami in služijo različnim odjemalcem. Da bi se soočili s temi izzivi, avtorji predstavijo distribuirano platformo COPE (Cloud Orchestration Policy Engine), ki omogoča ponudnikom oblaka izvajati deklarativno in avtomatizirano orkestracijo virov oblaka. Nasprotno mi definiramo dogodkovno-usmerjeno orkestracijsko arhitekturo za oblak, ki je odpornejša na napake in omogoča grafično kompozicijo ("drag-and-drop") virov in storitev oblaka, v obliki delovnih tokov. X. Liu et al. [226] predstavljajo splošno arhitekturo sistema za delovne tokove oblaka, ki odločajo, kako so sistemske komponente organizirane v različne sloje in kako le-te komunicirajo druga z drugo. Čeprav njihova arhitektura upošteva sloj aplikacije, platforme, virov ter fizični sloj, njihov pristop temelji izključno na orkestraciji virov oblaka, medtem ko mi vpeljujemo standardne zmožnosti orkestracije znotraj dogodkovno-usmerjene in kompenzacijske semantike.

Da bi naslovili iniciative za standardizacijo orkestracije virov oblaka, sta se pojavili dve ločeni specifikaciji: Topology and Orchestration Specification for Cloud Applications (TOSCA) iz OASIS [228] in ODCA Master Usage Model: Service Orchestration from Open Data Center Alliance (ODCA) [224]. Avtorji slednje specifikacije se osredotočajo na odkrivanje storitev (angl. Service Discovery) in orkestracijo nivoja IaaS. Specifikacija definira zahtevano avtomatizacijo za orkestracijo, ki vključuje programske vmesnike, vzorce interakcije, nadzorniške vmesnike in upravljanje z življenjskim ciklom. Naš predlog se osredotoča na vse storitvene modele računalniškega oblaka in vpeljuje principe upravljanja kompenzacij v orkestratorje oblaka. Na drugi strani, specifikacija TOSCA ponuja sredstva za opis topologije aplikacij za oblak, skupaj z njihovimi odvisnimi okolji, storitvami in artefakti znotraj enojne predloge storitve, ki omogoča razvijalcem postavljanje in upravljanje aplikacij, neodvisno od njihove infrastrukture in storitvenega modela. Kljub temu, se njihova specifikacija osredotoča samo na opisovanje aplikacijskega nivoja, medtem ko naše delo naslavlja storitve, ki se raztezajo preko celotnega sklada računalništva v oblaku in ponuja tako tehnično kot tudi poslovno orkestracijo oblaka. Še več, skupnost Atos Scientific Community [227] je izvedla raziskavo v obliki sistema PoC, ki je naslovil številne izzive in možne rešitve, da bi porazdelili poslovne procese preko več oblakov in na tak način raziskali

možnosti orkestracije v heterogenih okoljih oblaka. Ker so prednosti orkestracijskih orodij za heterogena okolja oblaka več kot očitne, je naš cilj razširiti zmožnosti orkestracijske platforme oblaka z vpeljavo zmožnosti upravljanja kompenzacij v arhitekturo delovnih tokov in nasloviti tako poslovno kot tudi tehnično orkestracijo. R. Ranjan et al. [229] predlagajo inovativno ogrodje Resource Orchestration Framework (ROF), ki uporablja virtualizacijsko platformo Java Widget ter Amazonove odprto-kodne API-je za omogočanje poenostavljene, intuitivne orkestracije virov in upravljanje s spletnimi aplikacijami. Čeprav ponujajo prijaznejšo in bolj dostopno orkestracijsko platformo za aplikacije oblaka, ne pokrivajo celotnega sklada oblaka in se ne osredotočajo na vpeljavo bolj odpornega sistema na napake, kot to počnemo v našem delu.

Sistemi znanstvenih delovnih tokov (angl. Scientific Workflow Systems) so pomembni za številne aplikacije, saj omogočajo kompozicijo in izvajanje kompleksne analize na distribuiranih virih [234]. Kljub temu da je bila paradigma delovnega toka izvorno vpeljana za avtomatizacijo poslovnih procesov, obstaja veliko interesa s strani znanstvenikov, da bi utilizirali tehnologije za avtomatizacijo kompleksnih, v večini primerov distribuiranih eksperimentov. Takšni tipi delovnih tokov so znani kot znanstveni delovni tokovi in so bolj kompleksni od poslovnih delovnih tokov, imajo posebne funkcionalnosti (npr. računska in podatkovna intenzivnost) ter manj interakcije s človekom. Pravzaprav so uporabljeni za simulacije, podatkovno analizo, procesiranje slik in številne druge funkcije. Najbolj odmevni primeri takšnih sistemov delovnih tokov so Pegasus Workflow Management System [231], Kepler Project [233] in Taverna Workflow Management System [232]. Čeprav so ti sistemi zmožni orkestrirati znanstvene aplikacije nad okoljem oblaka, ne vpeljujejo principov upravljanja kompenzacij in niso zmožni orkestrirati tako poslovnih kot tudi tehničnih delovnih tokov.

Primarni cilj kompenzacij je izvedba razveljavitvenih korakov, ki so že bili uspešno zaključeni, saj njihovi rezultati in morebiti stranski učinki niso več zaželeni in jih je potrebno povrniti v predhodno stanje. Koncept kompenzacij je že bil prisoten v industrijskih standardih, kot sta BPMN [64] in WS-BPEL [65]. Kljub temu, tehnologije poslovnih delovnih tokov niso bile izvorno zasnovane za orkestracijo delovnih tokov, specifičnih za oblak, in ponujajo podporo za modeliranje/izvajanje kompenzacij zgolj znotraj poslovnih delovnih tokov. V nasprotju s tem, mi oblikujemo tehnološko-neodvisno orkestracijsko arhitekturo, ki je specifična za oblak in podpira upravljanje kompenzacij v poslovni in tehnični domeni, ter predlagamo nov algoritem CAS za preiskovanje delovnih tokov orkestratorja oblaka in iskanje primerne aktivnosti za kom-

penzacijo.

Pregled sorodnih raziskav iz področja orkestracijske in kompenzacijske semantike je pokazal, da je model upravljanja kompenzacij v orkestracijskih platformah oblaka nov prispevek k znanosti, ki še ni bil obravnavan na podoben način.

"The best way to predict your future is to create it."
— Abraham Lincoln

Zaključek

7

Na podlagi izpostavljenih raziskovalnih problemov je bil glavni cilj disertacije predlagati referenčni model integracije infrastrukturnih in platformskih nivojev računalniškega oblaka ter v okviru tega modela definirati tri komplementarne modele (arhitekturni model IaaS, model obogatitve sodobnih izvajalnih platform s specifikami oblaka ter model upravljanja kompenzacij v orkestracijskih platformah oblaka). V doktorski disertaciji smo pokazali, da (1) arhitekturni model IaaS izboljša razumevanje nivoja IaaS in omogoči lažje sprejemanje tehnoloških odločitev, (2) model obogatitve sodobnih izvajalnih platform s specifikami oblaka omogoča administratorjem in razvijalcem aplikacij večji nadzor in boljše upravljanje aplikacij v oblaku ter (3) model upravljanja kompenzacij v orkestracijskih platformah oblaka povečuje učinkovitost pri oblikovanju robustnih delovnih tokov, ki so odpornejši na napake.

7.1 Sklepi

Kot del arhitekturnega modela IaaS (poglavje 3) smo predlagali enotno taksonomijo in arhitekturno ogrodje ter naredili evalvacijo sedmih platform IaaS. Namen taksonomije je bil identificirati in klasificirati fundamentalne komponente IaaS v urejene kategorije/sloje in uporabiti taksonomijo za oblikovanje arhitekturnega ogrodja IaaS. Taksonomijo sestavlja sedem ključnih slojev: sloj abstrakcije virov, sloj jedrnih storitev, podporni sloj, ki služi kot komunikacijski sloj med slojem jedrnih storitev in slojem abstrakcije virov, sloj varnosti, sloj upravljanja, nadzorniški sloj ter storitve z dodano vrednostjo. Preučili smo različne sisteme IaaS in jih nato z namenom ovrednotenja klasifikacije preslikali na taksonomijo. Nadaljevali smo z vpeljavo arhitekturnega ogrodja IaaS, ki temelji na enotni taksonomiji. Ogrodje enotne in celostne arhitekture IaaS je eden izmed ključnih znanstvenih doprinsov. Problem taksonomije ter arhitekturnega ogrodja IaaS še ni bil obravnavan na način, ki bi bil primerljiv našemu pristopu. S predlaganim arhitekturnim ogrodjem IaaS smo organizirali konceptualne sloje/komponente v arhitekturno ogrodje, ponudili skupno izhodišče za analizo, primerjavo in evalvacijo implementacijskih arhitektur IaaS ter definirali odvisnosti med posameznimi sloji in komponentami.

Po izvedbi celovite analize najpomembnejših komercialnih in odprto-kodnih produktov IaaS, tako v akademskem svetu, kot tudi v industriji smo evalvirali predlagano taksonomijo in arhitekturno ogrodje IaaS. Rezultati analize (poglavje 3.6) so pokazali funkcijsko podporo posamezne rešitve in indicirali, na kakšen način posamezna funkcionalnost ustreza taksonomiji in slojem/komponentam predlaganega arhitekturnega

ogrodja IaaS. Ustvarili smo bistveno primerjalno osnovo, ki omogoča organizacijam IT sprejemanje odločitev pri izbiri najustreznejše tehnologije. Evalvacija je pokazala opazno razliko funkcijske podpore med komercialnimi in odprto-kodnimi platformami IaaS, bistveno pomanjkljivost pomembnih arhitekturnih komponent v smislu izpolnjevanja obljube infrastrukturnih oblakov ter uporabnost predlagane taksonomije ter enotnega arhitekturnega ogrodja v realnem svetu, ki na tak način omogočata lažje sprejemanje odločitev v organizacijah IT pri izbiri najbolj ustrezne rešitve IaaS. Predlagana taksonomija in arhitekturno ogrodje IaaS sta bila evalvirana na številnih realnih projektih, vključno z KC OpComm, KC Class, SLA@SOI, Contrail ter na enem svetovalnem projektu za največjega telekomunikacijskega ponudnika v Sloveniji - Telekom. Pri vsakem projektu je bil izbran vsaj en ustrezen infrastrukturni oblak, kar kaže na uporabnost v realnem svetu ter na učinkovito validacijo arhitekturnega modela IaaS.

Računalništvo v oblaku spreminja način izgradnje in izvajanja tradicionalne programske opreme tako, da vpeljuje storitveno usmerjen (angl. Utility-Based) način dostave infrastrukture, platforme, aplikacij in storitev [55]. Jasno je, da je razvoj sistemskih arhitektur, ki lahko poenostavi in izboljša opravila visoko-nivojskega upravljanja in nadzora, ključnega pomena pri koriščenju potenciala oblaka [216]. Kljub temu je sprejetje te vizije še zmeraj ovirano zaradi pomanjkanja ustreznih tehnologij, saj obstoječe izvajalne platforme (npr. Java EE in .NET) niso bile zasnovane s specifikami oblaka.

Iz tega razloga smo zasnovali model obogatitve sodobnih izvajalnih platform tako, da smo definirali generičen sistem parametrov za izvajanje aplikacij v oblakih IaaS in PaaS (poglavje 4). Tak sistem preko politik in programskih direktiv transformira aplikacijske zahteve, ki so opredeljene s strani aplikacijskih administratorjev ali razvijalcev, v oskrbovanje in ostale operacije nad spodaj ležečo infrastrukturo oblaka. V tem primeru se lahko aplikacijski administrator/razvijalec osredotoči na aplikacijo in njeno postavitveno okolje, ne da bi pri tem moral biti še v vlogi sistemskega administratorja, administratorja podatkovne baze in administratorja vmesne programske opreme. Na tak način, naš predlog naslavlja vse deficitne zmožnosti obstoječih platform, ki temeljijo na vsebnikih, in jih vpeljuje na nivo aplikacije in platforme. Posledično smo strukturirali sistem parametrov v dve skupini: parametre za nadzor in parametre elastičnosti. Parametri za nadzor dodatno vpeljujejo tri parametre: parameter za spremljanje aplikacije, parameter za konfiguracijo lokacije ter parameter za kršitve SLA-jev. Skupina parametrov elastičnosti pa dodatno vpeljuje parameter za skaliranje poslovnega

sloja, analogen parameter za spletni sloj ter parameter za definiranje uporabniških pravila za elastičnost, ki temeljijo na metrikah relevantnih za celotno aplikacijo. Slednji parameter vpeljuje pristop, ki pri avtonomnem skaliranju kombinira stroške, kakovost in delovne obremenitve. Obe skupini parametrov predstavljata platformsko-neodvisne parametre, ki jih uporabljamo za preslikavo v platformsko-specifične programske direktive in politike (v našem primeru so parametri preslikani v Java EE Annotation Metadata). Preslikava med sistemom parametrov in sistemom metapodatkov Java EE je bila realizirana v obliki deklaracij novih anotacijskih tipov ter številnih razširitev postavitvenih deskriptorjev. Z evalvacijo smo pokazali, da preslikava sistema parametrov v platformo Java EE skrije kompleksnost spodaj ležeče infrastrukture oblaka, medtem ko ponuja administratorjem aplikacij in razvijalcev večji nadzor in lažje upravljanje njihovih aplikacij (tj. zmožnost programskega pridobivanja relevantnih metrik spremljanja, specificiranje lokacij izvajanja/postavitve določenega sloja aplikacije, programsko reagiranje na prekoračitve opozorilnega praga in praga kršitve, da bi izpolnili želene nivoje storitve ter definiranja politik za doseganje ročne ter samodejne elastičnosti).

Orkestracija oblaka vključuje medsebojno povezovanje razpršenih avtomatiziranih procesov (vključno s procesi odobritve) in virov IT, z uporabo delovnih tokov, ter ponujanje portala, preko katerega so ti delovni tokovi upravljani in nadzirani [235]. Čeprav je orkestracija oblaka hitro razvijajoče se raziskovalno področje, ne obstajajo dobro definirani standardi delovnih tokov, ki bi učinkovito kljubovali poslovnim in tehničnim kompleksnostim. Da bi omogočili zanesljivo orkestracijo delovnih tokov, ki je odporna na napake, je pomembno vzpostaviti učinkovit mehanizem upravljanja s kompenzacijami. Znotraj okolja računalniških oblakov obstaja naraščajoča tendenca po oblikovanju zahteve po IT v bolj poslovno-usmerjenih terminih, ki opisujejo dejanske zahtevane storitve. To omogoča močnejšo poravnavo IT-ja s poslovanjem in posledično povečuje pomembnost IT-ja v poslovnem svetu [99]. Takšna uvedba je lahko dosežena s ponujanjem robustne orkestracijske platforme, ki je odporna na napake, in omogoča avtomatizacijo storitev oblaka (od začetka, do konca) preko poslovne in tehnične domene.

Iz tega razloga smo vpeljali upravljanje kompenzacij v orkestracijsko arhitekturo oblaka tako, da smo predlagali generično rešitev, ki je lahko adaptirana s strani poljubnih izvajalnih pogonov za delovne tokove (poglavje 5). Da bi dosegli ta cilj, smo izvedli arhitekturno analizo obstoječih sistemov za delovne tokove specifične za oblak, definirali generični pristop za upravljanje kompenzacij, ki je aplikabilen nad orkestratorji

oblaka ter predlagali nov algoritem CAS za preiskovanje delovnih tokov orkestratorjev oblaka (tj. usmerjenih grafov) in iskanje primernih aktivnosti, ki jih je potrebno kompenzirati. Da bi dosegli dogodkovno-usmerjeno izboljšavo za upravljanje kompenzacij, smo predlagali podporo za upravljavce kompenzacij (angl. Compensation Handlers), ki vključujejo t. i. kompenzacijske dogodke tipa catch in kompenzacijske aktivnosti, ter kompenzacijske dogodke tipa throw. Pojem kompenzacijskega dogodka gnezdenih delovnih tokov (angl. Event Subworkflow) je ravno tako podprt.

Implementacija PoC je skupaj s primerom uporabe pokazala efektivnost in izvedljivost predlaganega pristopa. Primer uporabe je pokazal način preslikave predlagane generične arhitekture v izvršljive elemente BPMN 2.0. Pri tem upošteva razširjene aktivnosti delovnih tokov s specifikami oblaka. Implementacija PoC je pokazala:

1. da je generični delovni tok lahko transformiran v standardiziran izmenljivi format BPMN 2.0 in je potemtakem lahko uvožen in izvajan na poljubnem procesnem strežniku, ki je skladen z BPMN 2.0,

2. da so razširjeni procesi BPMN 2.0 lahko transformirani v generične delovne tokove, ki so nato lahko uporabljeni za avtomatizacijo opravil, specifičnih za oblak, in izvajanje algoritma CAS.

7.2 Prispevki k znanosti

Povzetek glavnih prispevkov k znanosti:

- Definicija referenčnega modela integracije infrastrukturnih in platformskih nivojev računalniškega oblaka, ki naslavlja integracijske izzive izvajalnih platform in infrastrukturnih oblakov, definira stične komponente oz. sinergije predlagane integracije, ter vpeljuje deficitne zmožnosti v krovni, integracijski model oblaka.

- Definicija arhitekturnega modela IaaS, ki ga sestavljata med seboj povezana taksonomija IaaS (poglavje 3.2) ter arhitekturno ogrodje IaaS (poglavje 3.3). Namen taksonomije je identificirati in klasificirati temeljne komponente IaaS v urejene kategorije/sloje ter uporabiti taksonomijo za oblikovanje arhitekturnega ogrodja IaaS. Namen arhitekturnega ogrodja IaaS je (1) organizirati konceptualne sloje/komponente v arhitekturno ogrodje, (2) ponuditi skupno izhodišče za analizo, primerjavo in evalvacijo implementacijskih arhitektur IaaS ter (3) definirati odvisnosti med posameznimi sloji in komponentami (poglavje 3.5).

- Definicija modela obogatitve sodobnih izvajalnih platform (tj. sistem parametrov) za izvajanje aplikacij v oblakih IaaS in PaaS, ki ga organiziramo v dve skupini: parametri za nadzor (poglavje 4.3.1) ter parametri za elastičnost (poglavje 4.3.2). Predlagan formalni sistem aplikacijske zahteve, ki so opredeljene s strani administratorja aplikacije ali razvijalca, transformira v operacije nad spodaj ležečimi viri IaaS (z uporabo politik in programskih direktiv).

- Preslikava abstraktnega modela parametrov v model metapodatkov za Java EE platformo, ki vključuje (1) deklaracijo novih anotacijskih metapodatkov, (2) razširitev globalnega postavitvenega deskriptorja (*application.xml*), (3) razširitev postavitvenega deskriptorja *ejb-jar.xml* in (4) razširitev postavitvenega deskriptorja *web.xml*. Predlagane deklaracije in razširitve naslavljajo aspekte elastičnosti ter nadzora aplikacij v oblaku.

- Vpeljava celovitega modela upravljanja kompenzacij v orkestratorje oblaka, pri čemer (1) izvedemo arhitekturno analizo obstoječih sistemov za delovne tokove specifične za oblak, (2) definiramo generični pristop za upravljanje kompenzacij, ki je aplikabilen nad orkestratorji oblaka, ter (3) predlagamo nov algoritem CAS za preiskovanje delovnih tokov orkestratorjev oblaka in iskanje primernih aktivnosti, ki jih je potrebno kompenzirati.

- Preslikava modela upravljanja kompenzacij orkestratorjev oblaka v razširjeno specifikacijo BPMN 2.0, ki vpeljuje nove tipe opravil s specifikami oblaka.

7.3 Prihodnje delo

Kot del prihodnjega dela načrtujemo nadaljnjo razširitev podpore za dogodkovno usmerjeno adaptacijo delovnih tokov v distribuiranem okolju oblaka. Pravzaprav je ta funkcionalnost pomembna zahteva orkestracijskih platform naslednje generacije, da bi reducirali nivo sklopljenosti med delovnimi tokovi in aktivnostmi zaradi manjših odvisnosti med aktivnostmi, in ponudili bolj fleksibilno kompozicijo aktivnosti, ki temelji na dogodkih, raje kot na zaporednih klicih skript. Za namene omogočanja učinkovitega odločanja v realnem času in doseganja boljše odzivnosti sistemov oblaka, se morajo delovni tokovi in aktivnosti odzivati ne samo na enostavne, ampak tudi kompleksne dogodke [236]. To postane očitno v sistemih oblaka naslednje generacije (angl. Future-Generation Cloud Systems), kjer tradicionalni izvori dogodkov niso samo lokalne apli-

kacije, storitve oblaka, VM-ji ali podatkovne baze, temveč tudi senzorji iz razpršenih, geo-redundantnih podatkovnih centrov, ki morajo biti upravljani in utilizirani.

"How terrible is wisdom when it brings no profit to the wise."
— Sophocles

LITERATURA

[1] L. Youseff, M. Butrico, and D. Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, 2008.

[2] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, 2009.

[3] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *Computer Communication Review*, 39(1): 50–55, 2009. Vaquero, Luis M. Rodero-Merino, Luis Caceres, Juan Lindner, Maik.

[4] Lizhe Wang, Marcel Kunze, Jie Tao, and Gregor von Laszewski. Towards building a cloud for scientific applications. *Adv. Eng. Softw.*, 42(9):714–722, 2011.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 10 2009.

[6] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing - the business perspective. *Decis. Support Syst.*, 51 (1):176–189, 2011.

[7] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. Scientific cloud computing: Early definition and experience. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, HPCC '08, pages 825–830, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3352-0. doi: 10.1109/HPCC.2008.38. URL http://dx.doi.org/10.1109/HPCC.2008.38.

[8] Lizhe Wang, Gregor Laszewski, Andrew Younge, He Xi, Marcel Kunze, Tao Jie, and Fu Cheng. Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146, 2010.

[9] Oracle Corporation. Oracle reference architecture - cloud infrastructure, November 2011. URL http://www.oracle.com/technetwork/topics/entarch/oracle-ra-cloud-infrastructure-r3-0-1395892.pdf.

[10] IBM Corporation. Ibm cloud computing reference architecture 2.0 - introduction and architecture overview, April 2011. URL http://www.opengroup.org/cloudcomputing/uploads/40/23840/CCRA.IBMSubmission.02282011.doc.

[11] Cisco Systems. Cloud: What an enterprise must know, Marec 2011. URL http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns836/ns976/white_paper_c11-617239.pdf.

[12] R.B. Bohn, J. Messina, Fang Liu, Jin Tong, and Jian Mao. Nist cloud computing reference architecture. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 594–596, July 2011. doi: 10.1109/SERVICES.2011.105.

[13] Distributed Management Task Force (DMTF). Architecture for managing clouds - a white paper from the open cloud standards incubator, Junij 2010. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf.

[14] Cloud Security Alliance (CSA). Security guidance for critical areas of focus in cloud computing, Marec 2011. URL https://cloudsecurityalliance.org/research/security-guidance/.

[15] Internet Engineering Task Force (IETF). Ieft cloud reference framework, Maj 2012. URL http://tools.ietf.org/html/draft-khasnabish-cloud-reference-framework-02.

[16] T. Tung. Defining a cloud reference model. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 598–603.

[17] Sun Yuanhui, Xiao Zongshui, Bao Dongmei, and Zhao Jie. An architecture model of management and monitoring on cloud services resources. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 3, pages V3–207–V3–211.

[18] N. Loutas, V. Peristeras, T. Bouras, E. Kamateri, D. Zeginis, and K. Tarabanis. Towards a reference architecture for semantically interoperable clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 143–150.

[19] National Institute of Standards and Technology (NIST), Marec 2014. URL http://www.nist.gov.

[20] Peter M. Mell and Timothy Grance. The nist definition of cloud computing, 2011.

[21] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[22] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S. Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Gener. Comput. Syst.*, 26(8):1226–1240, 2010.

[23] Gabriel Mateescu, Wolfgang Gentzsch, and Calvin J. Ribbens. Hybrid computing-where hpc meets grid and cloud computing. *Future Gener. Comput. Syst.*, 27 (5):440–453, 2011.

[24] Paul Marshall, Kate Keahey, and Tim Freeman. Improving utilization of infrastructure clouds. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 205–214, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4395-6. doi: 10.1109/CCGrid.2011.56. URL http://dx.doi.org/10.1109/CCGrid.2011.56.

[25] A. O. Garcia, S. Bourov, A. Hammad, J. van Wezel, B. Neumair, A. Streit, V. Hartmann, T. Jejkal, P. Neuberger, and R. Stotzka. The large scale data facility: Data intensive computing for scientific experiments. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1467–1474.

[26] Alexandre di Costanzo, Marcos Dias de Assuncao, and Rajkumar Buyya. Harnessing cloud technologies for a virtualized distributed computing infrastructure. *IEEE Internet Computing*, 13(5):24–33, 2009.

[27] Marcos Dias de Assuncao, Alexandre di Costanzo, and Rajkumar Buyya. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, pages 141–150, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-587-1. doi: 10.1145/1551609.1551635. URL http://doi.acm.org/10.1145/1551609.1551635.

[28] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, 53(4):535–545, 2009.

[29] M. Mahjoub, A. Mdhaffar, R.B. Halima, and M. Jmaiel. A comparative study of the current cloud computing technologies and offers, Nov 2011.

[30] Peter Sempolinski and Douglas Thain. A comparison and critique of eucalyptus, opennebula and nimbus, 2010.

[31] Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, and Qing Li. Comparison of several cloud computing platforms, Dec 2009.

[32] I. Voras, B. Mihaljevic, and M. Orlic. Criteria for evaluation of open source cloud computing solutions. In *Information Technology Interfaces (ITI), Proceedings of the ITI 2011 33rd International Conference on*, pages 137–142.

[33] S. Wind. Open source cloud computing management platforms: Introduction, comparison, and recommendations for implementation. In *Open Systems (ICOS), 2011 IEEE Conference on*, pages 175–179.

[34] T. Cordeiro, D. Damalio, N. Pereira, P. Endo, A. Palhares, G. Goncalves, D. Sadok, J. Kelner, B. Melander, V. Souza, and J. E. Mangs. Open source cloud computing platforms. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 366–371.

[35] M. Rodriguez-Martinez, J. Seguel, and M. Greer. Open source cloud computing tools: A case study with a weather application. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 443–449.

[36] D. Ogrizovic, B. Svilicic, and E. Tijan. Open source science clouds. In *MIPRO, 2010 Proceedings of the 33rd International Convention*, pages 1189–1192.

[37] Y. Ueda and T. Nakatani. Performance variations of two open-source cloud platforms. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–10.

[38] Li Ang, Yang Xiaowei, S. Kandula, and Zhang Ming. Comparing public-cloud providers. *Internet Computing, IEEE*, 15(2):50–53, 2011.

[39] Microsoft. .net framework, Januar 2014. URL http://www.microsoft.com/net.

[40] Oracle Corporation. Java ee technical documentation, Januar 2014. URL http://docs.oracle.com/javaee/.

[41] Microsoft Developer Network. Understanding service-oriented architecture, Januar 2004. URL http://msdn.microsoft.com/en-us/library/aa480021.aspx.

[42] Business process management (bpm), Marec 2014. URL http://www.bpm.com/.

[43] Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, and Chinmayee Srivathsa. *The Java EE 6 Tutorial: Basic Concepts*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2010. ISBN 0137081855, 9780137081851.

[44] M. Keen, R. Coutinho, S. Lippmann, S. Sollami, S. Venkatraman, S. Baber, H. Cui, C. Fleming, V.K.K. Gaddam, B. Hainey, et al. *Rational Application Developer for WebSphere Software V8 Programming Guide*. IBM redbooks. IBM Redbooks, 2011. ISBN 9780738435596. URL http://books.google.si/books?id=oBnCAgAAQBAJ.

[45] Oracle Corporation. Java ee annotation reference, Januar 2011. URL https://wikis.oracle.com/display/GlassFish/Java+EE+6+Annotation+Reference.

[46] Luis Rodero-Merino, Luis M. Vaquero, Eddy Caron, Adrian Muresan, and Frédéric Desprez. Building safe paas clouds: A survey on security in multitenant software platforms. *Computers & Security*, 31(1):96–108, 2012.

[47] JSR Community Expert Group JSRs: Java Specification Requests. Jsr 342: Java platform, enterprise edition 7 (java ee 7) specification, Maj 2013. URL http://www.jcp.org/en/jsr/detail?id=342.

[48] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1):45–52, 2011.

[49] JSR Community Expert Group JSRs: Java Specification Requests. Jsr 317: Javatm persistence 2.0, December 2009. URL http://jcp.org/en/jsr/detail?id=317.

[50] JSR Community Expert Group JSRs: Java Specification Requests. Jsr 221: Jdbc 4.0 api specification, Marec 2014. URL http://jcp.org/en/jsr/detail?id=221.

[51] Sla@soi - european fp7 project, Januar 2014. URL http://sla-at-soi.eu/.

[52] Reservoir - european fp7 project, Marec 2014. URL http://www.reservoir-fp7.eu/.

[53] Pankesh Patel, Ajith H. Ranabahu, and Amit P. Sheth. Service level agreement in cloud computing, December 2009. URL http://corescholar.libraries.wright.edu/knoesis/78.

[54] Changbin Liu, Boon Thau Loo, and Yun Mao. Declarative automated cloud resource orchestration, 2011. URL http://doi.acm.org/10.1145/2038916.2038942.

[55] J. Kirschnick, J. M. Alcaraz Calero, L. Wilcock, and N. Edwards. Toward an architecture for the automated provisioning of cloud services. *Communications Magazine, IEEE*, 48(12):124–131, 2010.

[56] Microsoft. System center 2012 service pack 1 (sp1) – orchestrator documentation, November 2013. URL http://download.microsoft.com/download/2/E/6/2E66C268-C3BD-4BCF-9D87-566A85ECF6EE/SC2012_Orch_CompleteDocumentation.pdf.

[57] Bahman Javadi, Martin Tomko, and Richard O. Sinnott. Decentralized orchestration of data-centric workflows in cloud environments. *Future Generation Computer Systems*, 29(7):1826–1837, 2013.

[58] Weimin Du, Jim Davis, Ming-chien Shan, and Umesh Dayal. Flexible compensation of workflow processes. 1997.

[59] VMware. Vmware vcenter orchestrator documentation, December 2012. URL http://www.vmware.com/support/pubs/orchestrator_pubs.html.

[60] BMC. Bmc atrium orchestrator, November 2012. URL http://www.bmc.com/products/product-listing/90902406-157022-1134.html.

[61] Cisco. Cisco process orchestrator, December 2012. URL http://www.cisco.com/en/US/products/ps11100/index.html.

[62] Hewlett-Packard. Hp operations orchestration, Januar 2014. URL http://www8.hp.com/us/en/software-solutions/operations-orchestration-it-process-automation/index.html.

[63] IBM. Ibm smartcloud orchestrator, Marec 2013. URL http://www-03.ibm.com/software/products/us/en/smartcloud-orchestrator/.

[64] OMG. Business process model and notation (bpmn 2.0), Maj 2013. URL http://www.omg.org/spec/BPMN/2.0/.

[65] OASIS. Web services business process execution language (ws-bpel 2.0), April 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[66] The YAWL Foundation. Yawl: Yet another workflow language, Junij 2013. URL http://www.yawlfoundation.org/.

[67] Microsoft. Windows workflow foundation (wf), Marec 2013. URL http://msdn.microsoft.com/en-us/vstudio/jj684582.aspx.

[68] Robert Dukaric and Matjaz B. Juric. Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems*, 29(5): 1196–1210, 2013.

[69] Robert Dukaric and Matjaz B. Juric. A taxonomy and survey of infrastructure-as-a-service systems. *Lecture Notes on Information Theory*, Vol.1(No.1):pp. 29–33, 2013.

[70] Robert Dukarić and Matjaž B. Jurič. Migracija obstoječih aplikacij na platforme za računalništvo v oblaku. *Uporabna informatika (Ljubljana)*, 19(3):136–146, 2011.

[71] Robert Dukarić and Matjaž B. Jurič. A unified architecture of iaas cloud solutions. In *Proceedings of the 1th International Conference on CLoud Assisted Service (CLASS Conference 2012)*, pages 28–34.

[72] Robert Dukarić and Matjaž B. Jurič. Migracija obstoječih rešitev v oblak. pages 1–11. Slovensko društvo Informatika, 18.-20. april 2011.

[73] Robert Dukarić and Matjaž B. Jurič. Prednosti in slabosti uporabe računalništva v oblaku v javni upravi [elektronski vir], 22.-23. november 2010.

[74] Robert Dukarić, Rajkumar Buyya, and Matjaž B. Jurič. Compensation handling in cloud orchestrators. *Poslano v objavo*, 2014.

[75] Robert Dukarić and Matjaž B. Jurič. Cloud-specific enrichment of contemporary container-based platforms. *Poslano v objavo*, 2014.

[76] SoftLayer Technologies. A brief history of cloud computing, Marec 2014.

[77] I. Foster, Zhao Yong, I. Raicu, and Lu Shiyong. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10.

[78] Gartner. Gartner highlights five attributes of cloud computing, December 2009. URL http://www.gartner.com/it/page.jsp?id=1035013.

[79] Drue Reeves. Cloud computing: Transforming it, December 2009. URL http://net.educause.edu/ir/library/pdf/ECR0901.pdf.

[80] Forrester Research. Supporting sustainable cloud services - investing in the network to deliver scalable, reliable, and secure cloud computing. URL http://www.thenewnetworkishere.com/pdfs/Forrester-Supporting_Sustainable_Cloud_Services.pdf,year={2009},.

[81] Cloud computing - definition. URL http://en.wikipedia.org/wiki/Cloud_computing.

[82] Vivek Kundra. Federal cloud computing strategy, Februar 2011. URL http://www.mail.governmenttrainingcourses.net/pdfs/Federal-Cloud-Computing-Strategy1.pdf.

[83] International Data Corporation (IDC). Is cloud computing right for you?, August 2010. URL http://www.qwest.com/qptcms/qCmsRepository/resources/pdfs/is-cloud-computing-right-for-you-idc-WP101305.pdf.

[84] Guilherme Galante and Luis Carlos E. de Bona. A survey on cloud computing elasticity, 2012.

[85] J. Van der Merwe, K. K. Ramakrishnan, M. Fairchild, A. Flavel, J. Houle, H. A. Lagar-Cavilla, and J. Mulligan. Towards a ubiquitous cloud computing infrastructure. In *Local and Metropolitan Area Networks (LANMAN), 2010 17th IEEE Workshop on*, pages 1–6.

[86] Amazon. Amazon virtual private cloud (amazon vpc), Februar 2014. URL http://aws.amazon.com/vpc/.

[87] jclouds, Januar 2013. URL http://jclouds.apache.org/.

[88] The Dasein Cloud API, Januar 2014. URL http://dasein-cloud.sourceforge.net/.

[89] Deltacloud, Februar 2014. URL http://deltacloud.apache.org/.

[90] M. Alhamad, T. Dillon, and E. Chang. Conceptual sla framework for cloud computing. In *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*, pages 606–610.

[91] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2007. URL http://www.ggf.org/Public_Comment_cs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf.

[92] Cisco Systems. Planning the migration of enterprise applications to the cloud - a guide to your migration options: Private and public clouds, application evaluation criteria, and application migration best practices, December 2010. URL https://www.cisco.com/en/US/services/ps2961/ps10364/ps10370/ps11104/Migration_of_Enterprise_Apps_to_Cloud_White_Paper.pdf.

[93] S. Kaisler and W. H. Money. Service migration in a cloud architecture. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–10.

[94] Lee Jae Yoo, Lee Jung Woo, Cheun Du Wan, and Kim Soo Dong. A quality model for evaluating software-as-a-service in cloud computing. In *Software Engineering Research, Management and Applications, 2009. SERA '09. 7th ACIS International Conference on*, pages 261–266.

[95] Paul Smyth. Cloud computing – a strategy guide for board level executives. kynetix technology group, Maj 2010. URL http://www.kynetix.com/media/19622/cloud_computing.pdf.

[96] N. Borenstein and J. Blake. Cloud computing standards: Where's the beef? *Internet Computing, IEEE*, 15(3):74–78, 2011.

[97] Balachandra Reddy Kandukuri, Ramakrishna Paturi V., and Atanu Rakshit. Cloud security issues, 2009. URL http://dx.doi.org/10.1109/SCC.2009.84.

[98] Usa patriot act., December 2001. URL http://www.fincen.gov/statutes_regs/patriot/index.html.

[99] G. Breiter and M. Behrendt. Life cycle and characteristics of services in the world of cloud computing. *IBM Journal of Research and Development*, 53(4): 3:1–3:8, 2009.

[100] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *IEEE Internet Computing*, 13(5):43–51, 2009.

[101] Gartner. Introducing the new magic quadrant for public cloud iaas, December 2011. URL http://blogs.gartner.com/lydia_leong/2011/12/13.

[102] Forrester. Market overview: Private cloud solutions (q2 2011), Maj 2011. URL www.forrester.com/go?docid=58924.

[103] Forrester. The state of infrastructure-as-a-service cloud standards, December 2011. URL www.forrester.com/go?docid=58065.

[104] Oracle Corporation. Achieving the cloud computing vision, Oktober 2010. URL http://www.oracle.com/technetwork/topics/entarch/architectural-strategies-for-cloud--128191.pdf.

[105] OpenStack. Open source software for building private and public clouds, Januar 2014. URL http://www.openstack.org/.

[106] libvirt. The virtualization api, Februar 2014. URL http://libvirt.org/.

[107] Citrix. Citrixxenserver management api, Marec 2014. URL http://docs.vmd.citrix.com/XenServer/6.0.0/1.0/en_gb/api/.

[108] VMware. vsphere api reference documentation, Januar 2010. URL https://www.vmware.com/support/developer/vc-sdk/visdk41pubs/ApiReference/.

[109] Microsoft. Windows management instrumentation (wmi), Januar 2014. URL http://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx.

[110] Amazon. Amazon simple storage service (amazon s3), April 2014. URL http://aws.amazon.com/s3/.

[111] John Bresnahan, Kate Keahey, David LaBissoniere, and Tim Freeman. Cumulus: an open source storage cloud for science. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 25–32, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0699-7. doi: 10.1145/1996109.1996115. URL http://dx.doi.org/10.1145/1996109.1996115.

[112] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[113] Apache. Hadoop distributed file system, Maj 2014. URL http://hadoop.apache.org/.

[114] XtreemFS. An open source distributed and replicated file system for the cloud, Marec 2014. URL http://www.xtreemfs.org/.

[115] Twisted storage, April 2014. URL http://twistedstorage.sourceforge.net/.

[116] Nimbus project, April 2014. URL http://www.nimbusproject.org/.

[117] Eucalyptus: The open source cloud platform, Marec 2014. URL http://open.eucalyptus.com/.

[118] Opennebula: The open source toolkit for data center virtualization, Januar 2014. URL http://opennebula.org/.

[119] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 124–131, May 2009. doi: 10.1109/CCGRID.2009.93.

[120] Amazon. Amazon elastic block store (ebs), Januar 2014. URL http://aws.amazon.com/ebs/.

[121] EMOTIVE Cloud. Elastic management of tasks in virtualized environments, Februar 2014. URL http://www.emotivecloud.net/.

[122] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Comput. Netw.*, 53(17):2923–2938, 2009.

[123] Tiago C. Ferreto, Marco A. S. Netto, Rodrigo N. Calheiros, and César A. F. De Rose. Server consolidation with migration control for virtualized data centers. *Future Gener. Comput. Syst.*, 27(8):1027–1034, 2011.

[124] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, and Stephen S. Yau. Efficient audit service outsourcing for data integrity in clouds. *Journal of Systems and Software*, 85(5):1083–1095, 2012.

[125] Advanced message queuing protocol (amqp), Januar 2014. URL http://www.amqp.org/.

[126] Activemq, Januar 2014. URL http://activemq.apache.org/.

[127] openqrm, Februar 2014. URL http://www.openqrm.com/.

[128] VMware. Vmware vcloud, Februar 2014. URL http://www.vmware.com/products/vcloud/.

[129] Microsoft. Microsoft private cloud, Februar 2014. URL http://www.microsoft.com/en-us/server-cloud/private-cloud/.

[130] SQLAlchemy. The python sql toolkit and object relational mapper, Februar 2014. URL http://www.sqlalchemy.org/.

[131] Luis M. Vaquero, Luis Rodero-Merino, and Daniel Morán. Locking the sky: a survey on iaas cloud security. *Computing*, 91(1):93–118, 2011.

[132] G. Peterson. Don't trust. and verify a security architecture stack for the cloud. *IEEE Security & Privacy*, 8 (5):83–86, 2010. Peterson, Gunnar.

[133] A. S. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy. Cloudsec: A security monitoring appliance for virtual machines in the iaas cloud model. In *Network and System Security (NSS), 2011 5th International Conference on*, pages 113–120.

[134] Cloud single sign-on & federated identity, April 2014. URL https://www.pingidentity.com/resource-center/SSO-and-Federated-Identity.cfm.

[135] Dana Petcu, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. Portable cloud applications—from theory to practice. *Future Generation Computer Systems*, (0), 2012.

[136] David Villegas, Norman Bobroff, Ivan Rodero, Javier Delgado, Yanbin Liu, Aditya Devarakonda, Liana Fong, S. Masoud Sadjadi, and Manish Parashar. Cloud federation in a layered service model. *J. Comput. Syst. Sci.*, 78(5):1330–1344, 2012.

[137] Rui Han, Moustafa M. Ghanem, Li Guo, Yike Guo, and Michelle Osmond. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Computer Systems*, 32(0):82–98, 2014.

[138] Apache. Tashi, Oktober 2012. URL http://incubator.apache.org/tashi/index.html.

[139] Ganglia monitoring system, Marec 2014. URL http://ganglia.sourceforge.net/.

[140] Nagios - the industry standard in it infrastructure monitoring, Marec 2014. URL http://www.nagios.org/.

[141] Haizea - an open source vm-based lease manager, December 2009. URL http://haizea.cs.uchicago.edu/index.html.

[142] Cisco. Cisco intelligent automation for cloud (ciac), September 2013. URL http://www.cisco.com/en/US/products/ps11869/index.html.

[143] BMC. Bmc cloud lifecycle management (clm), April 2014. URL http://www.bmc.com/products/product-listing/cloud-lifecycle-planning-management-software.html.

[144] Microsoft TechNet. System center 2012 r2 operations manager - operations manager key concepts, April 2014. URL http://technet.microsoft.com/en-us/library/hh230741.aspx.

[145] VMware. Vmware vcenter operations manager documentation, April 2014. URL https://www.vmware.com/support/pubs/vcops-pubs.html.

[146] Amazon. Amazon relational database service – high availability (multi-az), Februar 2013. URL http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html.

[147] Microsoft TechNet. Hyper-v overview, Januar 2014. URL http://technet.microsoft.com/en-us/library/hh831531.aspx.

[148] VMware. Vmware vcloud connector, Januar 2014. URL http://www.vmware.com/products/vcloud-connector/.

[149] Microsoft TechNet. System center 2012 app controller, Februar 2014. URL http://technet.microsoft.com/en-us/library/hh546834.aspx.

[150] Loutas Nikolaos, Kamateri Eleni, Bosi Filippo, and Tarabanis Konstantinos. Cloud computing interoperability: The state of play. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CLOUDCOM '11)*, volume 0, pages 752–757. IEEE Computer Society.

[151] Contrail project, Marec 2014. URL http://contrail-project.eu/.

[152] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 43–52, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4039-9. doi: 10.1109/CCGRID.2010.80. URL http://dx.doi.org/10.1109/CCGRID.2010.80.

[153] Katarzyna Keahey and Tim Freeman. Contextualization: Providing one-click virtual clusters. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ESCIENCE '08, pages 301–308, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3535-7. doi: 10.1109/eScience.2008.82. URL http://dx.doi.org/10.1109/eScience.2008.82.

[154] Gartner. Platform as a service (paas), Marec 2014. URL http://www.gartner.com/it-glossary/platform-as-a-service-paas/.

[155] Waratek. Java problems in the cloud: The need for multitenant java, April 2014. URL http://www.waratek.com/resources/whitepapers/java-problems-in-the-cloud.

[156] R. Ranjan, R. Buyya, and B. Benatallah. Special section: Software architectures and application development environments for cloud computing. *Software - Practice and Experience*, 42(4):391–394, 2012. Export Date: 25 February 2013 Source: Scopus.

[157] Antonio Goncalves. *Beginning Java EE 6 with GlassFish 3, Second Edition*. Apress, Berkely, CA, USA, 2nd edition, 2010. ISBN 143022889X, 9781430228899.

[158] Google. Google app engine (gae), April 2014. URL https://developers.google.com/appengine/.

[159] Microsoft. Windows azure: Microsoft's cloud platform, April 2014. URL http://www.windowsazure.com/en-us/.

[160] Amazon. Aws elastic beanstalk, April 2014. URL http://aws.amazon.com/elasticbeanstalk/.

[161] OASIS Committee Specification. Cloud application management for platforms version 1.1, Februar 2014. URL http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html.

[162] Daniel Beimborn, Thomas Miletzki, and Stefan Wenzel. Platform as a service (paas). *Business & Information Systems Engineering*, 3(6):381–384, 2011.

[163] Forrester. The forrester wave: Platform-as-a-service for vendor strategy professionals, Maj 2011. URL http://www.forrester.com/go?objectid=RES56295.

[164] Gartner. Paas 2012: Tactical risks and strategic rewards, Januar 2012. URL http://www.gartner.com/id=1886614.

[165] S. Dustdar, Y. Guo, B. Satzger, and Truong Hong-Linh. Principles of elastic processes. *Internet Computing, IEEE*, 15(5):66–71, 2011.

[166] M. P. Papazoglou and W. van den Heuvel. Blueprinting the cloud. *Internet Computing, IEEE*, 15(6):74–79, 2011.

[167] Tak Byung Chul, B. Urgaonkar, and A. Sivasubramaniam. Cloudy with a chance of cost savings. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1223–1233, 2013.

[168] Khalid Alhamazani, Rajiv Ranjan, Fethi Rabhi, Lizhe Wang, and Karan Mitra. Cloud monitoring for optimizing the qos of hosted applications. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 765–770.

[169] D. Bruneo. A stochastic model to investigate data center performance and qos in iaas cloud computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):560–569, 2014.

[170] H. Khazaei, J. Misic, and V. B. Misic. Performance analysis of cloud computing centers using m/g/m/m+r queuing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 23(5):936–943, 2012.

[171] B. Konig, J. M. Alcaraz Calero, and J. Kirschnick. Elastic monitoring framework for cloud infrastructures. *Communications, IET*, 6(10):1306–1315, 2012.

[172] Tram Truong Huu, Guilherme Koslovski, Fabienne Anhalt, Johan Montagnat, and Pascale Vicat-Blanc Primet. Joint elastic cloud and virtual network framework for application performance-cost optimization. *J. Grid Comput.*, 9(1):27–47, 2011.

[173] VMware. vcenter chargeback manager documentation, December 2013. URL http://www.vmware.com/support/pubs/vcbm_pubs.html.

[174] Amazon. Amazon ec2 spot instances, Marec 2014. URL http://aws.amazon.com/ec2/spot-instances/.

[175] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1):57–81, March 2003. ISSN 1064-7570. doi: 10.1023/A:1022445108617. URL http://dx.doi.org/10.1023/A:1022445108617.

[176] Vincent C. Emeakaroha, Marco A. S. Netto, Rodrigo N. Calheiros, Ivona Brandic, Rajkumar Buyya, and César A. F. De Rose. Towards autonomic detection of sla violations in cloud infrastructures. *Future Gener. Comput. Syst.*, 28(7):1017–1029, 2012.

[177] Oracle Corporation. Datasource objects and connection pools, Avgust 2013. URL http://docs.oracle.com/javaee/6/tutorial/doc/bncjj.html.

[178] Microsoft. Ado.net., Marec 2014. URL http://msdn.microsoft.com/en-us/library/e80y5yhx.aspx.

[179] Vijindra and Sudhir Shenai. Survey on scheduling issues in cloud computing. *Procedia Engineering*, 38 (0):2881–2888, 2012.

[180] VMware. Vmware vcloud architecture toolkit cloud bursting (vcat 3.1.2), Marec 2014. URL http://www.vmware.com/support/pubs/orchestrator_pubs.html.

[181] Microsoft. Msdn – developer network. attributes (c# and visual basic), December 2013. URL http://msdn.microsoft.com/en-us/library/z0w1kczw.aspx.

[182] Python Wiki. Decorators for functions and methods, Junij 2003. URL http://legacy.python.org/dev/peps/pep-0318/.

[183] Oracle Corporation. Sun java system application server platform edition 9 application deployment guide (chapter 1 assembling and deploying applications), December 2010. URL http://docs.oracle.com/cd/E19501-01/819-3660/beact/index.html.

[184] Distributed Management Task Force (DMTF). Open virtualization format (ovf) specification, December 2012. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0.pdf.

[185] JBoss Community. Jboss application server 7, Marec 2014. URL http://jbossas.jboss.org/.

[186] Hector Garcia-Molina and Kenneth Salem. Sagas. *SIGMOD Rec.*, 16(3):249–259, 1987.

[187] A. Kertesz, G. Kecskemeti, and I. Brandic. An interoperable and self-adaptive approach for sla-based service virtualization in heterogeneous cloud environments. *Future Generation Computer Systems*, 32(0): 54–68, 2014.

[188] Gwan-Hwan Hwang, Yung-Chuan Lee, and Bor-Yih Wu. *A New Language to Support Flexible Failure Recovery for Workflow Management Systems*, volume 2806 of *Lecture Notes in Computer Science*, chapter 12, pages 135–150. Springer Berlin Heidelberg, 2003.

[189] Nina Edelweiss and M. Nicolao. Workflow modeling: exception and failure handling representation. *Computer Science, 1998. SCCC '98. XVIII International Conference of the Chilean Society of*, pages 58 – 67, 1998.

[190] C. Hagen and G. Alonso. Exception handling in workflow management systems. *Software Engineering, IEEE Transactions on*, 26(10):943–958, 2000.

[191] T. Takemura. Formal semantics and verification of bpmn transaction and compensation. In *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pages 284–290.

[192] Microsoft Developer Network. Compensation vs. transactions, Avgust 2012. URL http://msdn.microsoft.com/en-us/library/dd489432.aspx.

[193] Forrester. Cloud orchestration models, 2011. URL http://www.forrester.com/Cloud+Orchestration+Models/fulltext/-/E-RES58375?objectid=RES58375.

[194] Forrester. Five trends that will change saas sourcing, Februar 2011. URL http://www.forrester.com/Five+Trends+That+Will+Change+SaaS+Sourcing/fulltext/-/E-RES60428.

[195] International Data Corporation (IDC). Idc marketscape: Worldwide distributed server/workload automation software 2012 vendor analysis, Julij 2012. URL https://ssl.www8.hp.com/ww/en/secure/pdf/4aa4-4358enw.pdf.

[196] Gregory Mentzas, Christos Halaris, and Stylianos Kavadias. Modelling business processes with workflow systems: an evaluation of alternative approaches. *International Journal of Information Management*, 21(2): 123–135, 2001.

[197] The Workflow Management Coalition (WfMC) Specification. The workflow reference model, Januar 1995. URL http://www.wfmc.org/standards/docs/tc003v11.pdf.

[198] Microsoft TechNet. Runbook concepts, November 2013. URL http://technet.microsoft.com/en-us/library/hh403820.aspx.

[199] Michele Chinosi and Alberto Trombetta. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[200] OMG. Unified modeling language (uml) – infrastructure specification, Avgust 2011. URL http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF.

[201] OMG. Unified modeling language (uml) – superstructure specification, Avgust 2011. URL http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF.

[202] OMG. Xml metadata interchange (xmi), December 2007. URL http://www.omg.org/spec/XMI/2.1.1/.

[203] Microsoft TechNet. Windows powershell scripting, Oktober 2013. URL http://technet.microsoft.com/en-us/scriptcenter/dd742419.aspx.

[204] VMware. Vmware vsphere powercli documentation, Februar 2014. URL https://www.vmware.com/support/developer/PowerCLI/.

[205] Ales Frece and Matjaz B. Juric. Modeling functional requirements for configurable content- and context-aware dynamic service selection in business process models. *Journal of Visual Languages & Computing*, 23 (4):223–247, 2012.

[206] JBoss Community. jbpm, Marec 2014. URL http://www.jboss.org/jbpm/.

[207] IBM. Business process management (bpm), Marec 2014. URL http://www-03.ibm.com/software/products/us/en/category/BPM-SOFTWARE.

[208] B. P. Rimal, Choi Eunmi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 44–51.

[209] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011.

[210] Forrester. The evolution of cloud computing markets, Julij 2010. URL www.forrester.com/go?docid=57232.

[211] OpenCrowd. Cloud computing vendors taxonomy, December 2013. URL http://cloudtaxonomy.opencrowd.com/.

[212] Intel Corporation. Cloud computing taxonomy and ecosystem analysis, Marec 2010. URL https://communities.intel.com/docs/DOC-4932.

[213] Luis M. Vaquero, Daniel Morán, Fermín Galán, and Jose M. Alcaraz-Calero. Towards runtime reconfiguration of application control policies in the cloud. *J. Netw. Syst. Manage.*, 20(4):489–512, 2012.

[214] S. Dustdar, Guo Yike, Han Rui, B. Satzger, and Truong Hong-Linh. Programming directives for elastic computing. *Internet Computing, IEEE*, 16(6): 72–77, 2012.

[215] Rodrigo N. Calheiros, Adel Nadjaran Toosi, Christian Vecchiola, and Rajkumar Buyya. A coordinator for scaling elastic applications across multiple clouds. *Future Generation Computer Systems*, 28(8):1350–1362, 2012.

[216] S. A. De Chaves, R. B. Uriarte, and C. B. Westphall. Toward an architecture for monitoring private clouds. *Communications Magazine, IEEE*, 49(12):130–137, 2011.

[217] J. Spring. Monitoring cloud computing by layer, part 1. *Security & Privacy, IEEE*, 9(2):66–68, 2011.

[218] P. Hasselmeyer and N. d"Heureuse. Towards holistic multi-tenant monitoring for virtual data centers. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pages 350–356.

[219] M. Anand. Cloud monitor: Monitoring applications in cloud. In *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*, pages 1–4.

[220] M. Dhingra, J. Lakshmi, and S. K. Nandy. Resource usage monitoring in clouds. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 184–191.

[221] Deqing Zou, Wenrong Zhang, Weizhong Qiang, Guofu Xiang, Laurence Tianruo Yang, Hai Jin, and Kan Hu. Design and implementation of a trusted monitoring framework for cloud platforms. *Future Generation Computer Systems*, 29(8):2092–2102, 2013.

[222] O. Adinolfi, R. Cristaldi, L. Coppolino, and L. Romano. Qos-monaas: A portable architecture for qos monitoring in the cloud. In *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, pages 527–532.

[223] Rodrigo N. Calheiros, Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya. The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. *Future Gener. Comput. Syst.*, 28(6):861–870, 2012.

[224] Open Data Center Alliance. Master usage model: Service orchestration rev 1.0, November 2012. URL http://www.opendatacenteralliance.org/docs/ODCA_Service_Orch_MasterUM_v1.0_Nov2012.pdf.

[225] Yun Mao, Changbin Liu, Jacobus E. van der Merwe, and Mary Fernandez. Cloud resource orchestration: A data-centric approach. pages 241–248. www.crdrdb.org, 2011.

[226] Xiao Liu, Dong Yuan, Gaofeng Zhang, Wenhao Li, Dahai Cao, Qiang He, Jinjun Chen, and Yun Yang. *Cloud Workflow System Architecture*, chapter 2, pages 13–18. SpringerBriefs in Computer Science. Springer New York, 2012.

[227] Atos. Cloud orchestration: A real business need, Oktober 2011. URL http://nl.atos.net/content/dam/nl/documents/atos-wp-cloud-orchestration.pdf.

[228] OASIS. Topology and orchestration specification for cloud applications (tosca), Januar 2013. URL http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf.

[229] Rajiv Ranjan, Boualem Benatallah, and Mingyi Wang. *A Cloud Resource Orchestration Framework for Simplifying the Management of Web Applications*, volume 7221 of *Lecture Notes in Computer Science*, chapter 33, pages 248–249. Springer Berlin Heidelberg, 2012.

[230] Amazon. Amazon simple workflow service (swf), Marec 2014. URL http://aws.amazon.com/swf/.

[231] Pegasus workflow management system, Januar 2014. URL http://pegasus.isi.edu/.

[232] Kepler project, Februar 2014. URL https://kepler-project.org/.

[233] Taverna workflow management system, Januar 2014. URL http://www.taverna.org.uk/.

[234] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[235] Network Computing. Private cloud automation, orchestration, and measured service, Junij 2011. URL http://www.networkcomputing.com/private-cloud-tech-center/private-cloud-automation-orchestration-a/231000293?pgno=1.

[236] M. Potocnik and M. Juric. Towards complex event aware services as part of soa. *Services Computing, IEEE Transactions on*, PP(99):1–1, 2013.