DevOps okruženje na Google Cloud platformi

Jakab, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

DEVOPS OKRUŽENJE NA GOOGLE CLOUD PLATFOTRMI

Diplomski rad

Ivan Jakab

Mentor:
Zdravko Krpić

Osijek, 2021. godina

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 25.08.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Ivan Jakab
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1059R, 06.10.2019.
OIB studenta:	74955763465
Mentor:	Doc.dr.sc. Zdravko Krpić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Bruno Zorić
Član Povjerenstva 1:	Doc.dr.sc. Zdravko Krpić
Član Povjerenstva 2:	Izv. prof. dr. sc. Ivica Lukić
Naslov diplomskog rada:	DevOps okruženje na Google Cloud platformi
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu rada potrebno je opisati postupak razvoja programske podrške na cloud platformama te ih usporediti s rješenjima upotrebom najmenskih poslužitelja. Također, potrebno je opisati osnovna načela DevOps načina razvoja programske podrške, a zatim se usredotočiti na DevOps način razvoja uz potporu cloud platformi, opisati alate koje one nude i kako se integriraju u postupku razvoja digitalnih proizvoda. Opisati karakteristike "deployment" procesa za navedenu platformu te navesti i opisati tehnologije koje nude cloud platforme (s naglaskom na Google Cloud) za spremište izvornog koda, izradu (build) programskog koda, izvršavanje programskog koda, pohranu
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	25.08.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 08.09.2021.

Ime i prezime studenta:	Ivan Jakab
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1059R, 06.10.2019.
Turnitin podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: DevOps okruženje na Google Cloud platformi**

izrađen pod vodstvom mentora Doc.dr.sc. Zdravko Krpić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Ivan Jakab_____, OIB: __74955763465_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Diplomski sveučilišni studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _DevOps okruženje na Google Cloud platformi**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 08.09.2021._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1 UVOD 1

2 PREGLED PODRUČJA TEME 3

3 DEVOPS I IZAZOVI DEVOPS OKRUŽENJA 6

3.1 Problemi čestog postavljanja programskog koda 7

3.2 Problemi skaliranja računalnih resursa 7

3.3 Opis principa DevOps okruženja 8

3.4 Opis izrade DevOps okruženja 9

4 TEHNOLOGIJE IZVEDBE EKSPERIMENTA 11

4.1 Docker 11

4.2 Google platforma u oblaku 11

4.2.1 Repozitoriji programskog koda i docker slika 12

4.2.2 Cloud SQL 12

4.2.3 Cloud Build 12

4.2.4 Google Cloud Run 13

4.2.5 Cloud CDN i Cloud Armor 14

5 IZRADA DEVOPS OKRUŽENJA SUSTAVA ZA PRAĆENJE UREDSKOG POSLOVANJA 15

5.1 Dizajn baze podataka 16

5.2 Tehnologije i arhitektura programskog koda aplikacije 16

5.3 Krajnje točke 17

5.4 Izrada DevOps okruženja aplikacije 19

5.4.1 Izrada docker slike 20

5.4.2 Postavljanje slike na Cloud Run 22

5.4.3 Praćenje sustava 23

5.4.4 Mogućnosti unaprjeđenja DevOps okruženja 25

6 ZAKLJUČAK....26
LITERATURA....27
SAŽETAK....28
ABSTRACT....28
ŽIVOTOPIS....29

1 UVOD

Porast suvremenog tehnološkog napretka sa sobom nužno povlači veću potrebu za računalnim resursima. Sakupljanje analitičkih podataka, visok promet, prikupljanje i analiza podataka koji pristižu velikom brzinom sve su češći zahtjevi modernih aplikacija. Uz povećane računalne zahtjeve, u suvremenim razvojnim timovima javlja se i potreba za čestom isporukom novog programskog koda, obično na tjednoj bazi.

Iz porasta zahtjeva proizlaze osnovna dva problema kojima će se ovaj rad baviti – skaliranje računalnih resursa i pouzdan proces kontinuiranog postavljanja novog programskog koda. Ova se dva problema opisuju zajedno jer su često integrirana – javlja se potreba za okruženjem u kojem se na jednostavan način može pouzdano postaviti novi programski kod, a računalna arhitektura na koju se programski kod postavlja treba moći pogoniti programsko rješenje. Budući da se taj proces izvršava kontinuirano, često se upotrebljava pojam kontinuirana integracija, a okruženje u kojem je izvedeno razvojno i operacijsko (*engl. Development Operations, DevOps*) okruženje.

Kako bi se ispunili programski zahtjevi prilikom porasta broja korisnika i opterećenja, potrebno je skalirati računalne resurse. Prilikom skaliranja računalnih resursa dolazi do brojnih problema - manjak prostora za pohranu, kvarovi, konstrukcija mreže i slično. Broj ljudi koji to može kvalitetno održavati je malen, a cijena je visoka. Uz to, ako proces postavljanja novoga programskog koda na platformu nije pouzdan i jednostavan, nužno dolazi do brojnih ljudskih grešaka i velikog gubitka vremena, osobito ako je proces ručni. Kada se novi programski kod postavi, čest je problem nedostatak alata za praćenje novog ponašanja – prikaz grešaka, prikaz iskorištenja računalnih resursa, prikaz prometa i slično. To često uzrokuje duže vrijeme do popravka greške.

Za rješavanje problema skaliranja računalnih resursa, često se koriste platforme za računarstvo u oblaku te postoje razna rješenja za kontinuiranu integraciju koji integriraju programski kod s tim platformama. Poznata su rješenja Gitlab Auto DevOps, Github Actions, Google App Engine, JetBrains TeamCity, CircleCI Orbs. Osnovni nedostaci koji se javljaju u ovim rješenjima su ručna integracija s računalnom platformom krajnje aplikacije, održavanje vlastitih računalnih resursa za izvršavanje procesa i nedostatak mogućnosti prilagodbe.

U ovom radu naglasak će biti na jednostavnu kontinuiranu integraciju u platformu za računarstvo u oblaku. Poseban će naglasak biti stavljen na jednostavno skaliranje računalnih

resursa pri izvršavanju krajnje aplikacije i izvođenju procesa postavljanja novog programskog koda na platformu. Uz to, naglasak će biti stavljen i na jednostavnost, u vidu toga da nema potrebe za održavanjem vlastitih računalnih resursa i kompleksne konfiguracije.

Cilj rada je izraditi okruženje za demonstrativnu aplikaciju takvo da se novi programski kod jednostavno postavlja na platformu za računarstvo u oblaku kada se dogodi promjena programskog koda te da platforma može automatski skalirati sve svoje dijelove s porastom opterećenja. Potom je potrebno osigurati dobar sustav praćenja novonastalog ponašanja. To sve je potrebno izvesti bez direktne interakcije s fizičkim ili virtualnim poslužiteljima i uz minimalnu količinu konfiguracije.

U poglavlju 2 bit će predstavljena trenutno poznata rješenja za problematike kojima će se rad baviti. U poglavlju 3 detaljnije će biti objašnjeni česti problemi koji se javljaju prilikom razvoja suvremenih aplikacija, kao i njihova rješenja u vidu principa i ciljeva DevOps okruženja. Poglavlje 4 sadržavat će detaljniji opis tehnologija koje se koriste za lakšu implementaciju DevOps okruženja. Konačno, u poglavlju 5 opisat će se izrada demonstrativne aplikacije za praćenje uredskog poslovanja, izrada DevOps okruženja za tu aplikaciju te će biti opisan detaljan tehnički pregled izrade te aplikacije.

2 PREGLED PODRUČJA TEME

Ideja DevOpsa i kontinuirane integracije već je neko vrijeme prisutna u industriji, u kojoj je razvijena u gotovo svim većim kompanijama. Nešto je slabije prisutna u znanstvenoj literaturi, jer je popularna tek nekoliko godina. Kompanije detalje implementacije ne drže javno dostupnima, no mogu se istražiti popularna otvorena rješenja i najbolji procesi ovog područja. Osnovna je zadaća kontinuirane integracije pružiti jednostavno okruženje za kontinuirano postavljanje novog programskog koda na računalnu infrastrukturu, automatsko skaliranje računalnih resursa, automatsko pokretanje testova ili dijagnostike programskog koda, kontinuirano praćenje ponašanja aplikacije i slično. Svako rješenje opisano u danjem tekstu nužnu mora pružiti mogućnost postavljanja novog programskog koda na računalnu infrastrukturu.

Prvu skupinu rješenja predstavljaju alati direktno integrirani u platforme upravljanja programskim kodom. Najpoznatiji su Gitlab Auto DevOps opisan u [1] i Github Actions opisane u [2]. One su izrađene na platformama koje primarno služe dijeljenju programskog koda, a to je pogodno jer je praćenje promjena programskog koda ključni dio kontinuirane integracije. Drugu skupinu rješenja čine rješenja izrađena na platformama za automatizaciju procesa, a predstavnici su JetBrains TeamCity kojeg su autori objasnili u [3] i CircleCI Orbs opisan u [4]. Budući da je ova skupina izrađena na platformama za automatizaciju, ova rješenja pružaju bolje alate za izradu automatiziranih procesa, no moraju se dodatno konfigurirati kako bi imala pristup programskom kodu i platformi za izvršavanje krajnje aplikacije. Konačno, posljednja skupina su već gotove platforme, koje pružaju cijela integrirana okruženja za izradu i izvršavanje aplikacije – način za postavljanje programskog koda, mreža, pohrana, baza podataka i slično. Jedan predstavnik ovakvog okruženja je Google App Engine opisan u [5]. U tablici 2.1 dodatno su navedene prednosti i mane pojedinih opisanih skupina rješenja.

Tablica 2.1 Tablični prikaz prednosti i mana pojedinih skupina postojećih rješenja.

Skupina rješenja	Prednosti	Mane
Platforme za upravljanje programskim kodom	Direktno integrirano s programskim kodom	Manji set alata za izradu automatiziranih procesa
Platforme za automatizaciju procesa	Najveći set alata za izradu automatiziranih procesa	Potrebno je ručno integrirati programski kod i infrastrukturu
Integrirana okruženja za izradu i izvršavanje aplikacije	Programski kod i infrastruktura automatski su integrirani	Manji set alata za izradu automatiziranih procesa

Radi bližeg pregleda prve kategorije, dodatno će se opisati Gitlab Auto DevOps, jer je kompleksniji od *Github Actions*. Iz druge kategorije dodatno će se objasniti JetBrains TeamCity radi veće korištenosti. Konačno, dodatno će se objasniti i Google App Engine kako bi se pobliže opisali prednosti i nedostaci treće skupine rješenja.

Česta je upotreba Gitlab platforme, kao osnove DevOps okruženja. Ona je posebice pogodna jer integrira programski kod, prava pristupa, i automatizirane procese, koji se mogu izvršiti kada se dogode određeni događaji. Gitlab Auto DevOps pruža već definirano okruženje, koje je moguće konfigurirati autorizacijskim podacima. Međutim, nedostatak ovog pristupa jest u tome da se resursi za izvršavanje procesa ručno konfiguriraju i usklađuju s platformom, što dovodi do problema sa skaliranjem, čestih padova i izgubljenog vremena. U ovom se radu procesi izvršavaju na platformi u oblaku i nema potrebe za konfiguriranjem vlastitih poslužitelja niti problema sa skaliranjem. Dodatno, problem je što *Gitlab* platforma ne nudi računalne resurse, pa se mora ručno konfigurirati pristup platformi na kojoj se programski kod izvršava. U ovom se radu koristi jedna platforma i za izvršavanje krajnje aplikacije i za pokretanje automatiziranih procesa, zbog čega nema dodatne konfiguracije radi integriranja platformi.

JetBrains TeamCity nudi gotovo okruženje za postavljanje programskog koda, pokretanje automatiziranih testova, analizu kvalitete programskog koda i slično. Ovo okruženje je napravljeno na platformi zamišljenoj za automatizaciju procesa, pa zato ima više mogućnosti u odnosu na Gitlab Auto DevOps – glavna je prednost u tome što nudi računalne resurse za izvođenje automatiziranih procesa u oblaku, ne moraju se konfigurirati niti održavati ručno. Međutim, ta platforma ne integrira repozitorij programskog koda, pa se pristup programskom kodu mora ručno konfigurirati, kao i sustav na kojem bi se aplikacija izvršavala. U ovom radu DevOps okruženje je napravljeno na platformi koja integrira programski kod, platformu za automatizaciju procesa i platformu za pokretanje aplikacije pa je potrebna konfiguracija tako znatno smanjena.

Konačno, Google App Engine je platforma koja nudi integrirano okruženje za razvoj i izvođenje aplikacija. Integrira repozitorij programskog koda, postavljanje novog programskog koda prilikom promjene i računalne resurse u oblaku računala. Glavni nedostatak ovog rješenja je u tome što je vrlo usko, podržava samo određene tehnologije u kojima krajnja aplikacija može biti napisana i nije moguće prilagoditi automatizirane procese postavljanja programskog

koda. U ovom se radu aplikacije postavljaju u formatu koji je neovisan o tehnologiji izrade i moguće je izvesti bilo kakve naredbe prilikom postavljanja programskog koda.

3 DEVOPS I IZAZOVI DEVOPS OKRUŽENJA

Prema [6], DevOps dolazi od spoja dvije engleske riječi, *Development* i *Operations*. To daje naznaku da se radi o uskom spoju između operacija (zadaci poput izrade računalne arhitekture, operacijskog sustava, pokretanje procesa na poslužitelju, mreža i komunikacija poslužitelja s okolinom, postavljanje vatrozida i slično) i samog razvoja aplikacije. Ideja je pronaći najbolju infrastrukturu za danu aplikaciju, postaviti odgovarajuće okruženje, pružiti jednostavno sučelje za novog postavljanje programskog koda, te raditi na potrebnom skaliranju računalnih resursa.

Računalna infrastruktura i način postavljanja novog programskog koda usko su vezani uz razvoj i način izrade aplikacije. Glavna je zadaća DevOps okruženja pružiti jednostavno sučelje za kontinuiranu integraciju programskog koda s platformom na kojoj se krajnja aplikacija izvršava i pratećom infrastrukturom, a zatim pružiti dobro sučelje za praćenje promjena, administraciju i analizu sustava, osmisliti dobar način za simuliranje produkcijske okoline za testiranje i slično. Takvo okruženje treba biti izrađeno za konkretne potrebe same aplikacije i u uskoj suradnji s razvojnim timom.

Problemi koji se javljaju u prilikom razvoja modernih aplikacija zbog kojih je preporučljivo razviti kvalitetno DevOps okruženje su:

- Česta potreba za postavljanjem novog programskog koda, a povezani problemi su
    - Potreba za postavljanjem lokalnog razvojnog okruženja
    - Potreba za postavljanje testnog ili prezentacijskog razvojnog okruženja
- Problemi skaliranja računalnih resursa
- Otežano praćenje ponašanja krajnje aplikacije
- Otežano detektiranje i uklanjanje problema koji se jave u produkcijskom okruženju

Prve dvije stavke će biti dodatno objašnjene u danjem tekstu jer su to osnovne zadaće kvalitetnog DevOps okruženja koje imaju najveći utjecaj na razvoj i ispravan rad krajnje aplikacije, a ispravna implementacija stavki će biti osnova eksperimentalnog dijela ovog rada.

3.1 Problemi čestog postavljanja programskog koda

U suvremenim razvojnim okruženjima sve je češća potreba za postavljanjem i isporukom novoga programskog koda. To se događa jer se suvremene aplikacije sve češće isporučuju u obliku manjih stabilnih verzija s relativno malim izmjenama. Primjerice, kompanija može imati potrebu postaviti novu verziju aplikacije na tjednoj bazi. Uz to, vjerojatno uz produkcijsko postoji i razvojno okruženje, a ako projekt ima dovoljan budžet može biti i testno, prezentacijsko, predprodukcijsko i slično. Još se na to može nadodati postavljanje lokalnog okruženja kod razvojnog tima, kao i mogućnost da se kompletna aplikacija sastoji od više manjih neovisnih servisa.

Ako u ovako opisanoj situaciji proces postavljanja okruženja nije jednostavan, efikasan i pouzdan, razvojni tim počinje gubiti sve više vremena na postavljanje novog programskog koda. Uz to, ako je posao sklon ljudskim greškama (primjerice, radi se ručnim spajanjem na poslužitelja na kojem se ručno izrađuje i pokreće krajnja aplikacija) često će se događati pogreške. Potom, česti su problemi da su na različitim operacijskim sustavima neki servisi drugačije implementirani ili ne rade, poput baze podataka.

3.2 Problemi skaliranja računalnih resursa

Osim problema s postavljanjem programskog koda, učestali su i problemi sa skaliranjem računalnih resursa. Aplikacija često može „prividno“ raditi, odnosno raditi pri niskom opterećenju u kontroliranom okruženju. No, kada se gotov digitalni proizvod sastoji od velikog broja manjih međuovisnih dijelova, a opterećenje se povisi, postaje jasno da infrastruktura ne može podržati potrebne zadatke.

Osnovni problem je u međuovisnosti – ako jedan dio ukupnog proizvoda nije u funkciji, ukupan proizvod nije u funkciji. Raspoloživost ukupnog proizvoda tako brzo opada dodavanjem međuovisnih dijelova. Kako bi se ovo izbjeglo, nužno je uvesti redundanciju svih ključnih dijelova ukupnog sustava, što je često jako težak pothvat za izvesti samostalno, ili gotovo nemoguć ako je redundancija treba biti globalna. Kvalitetno i pouzdano postavljanje više instanci servisa na različita računala, koji komuniciraju i balansiraju opterećenje je ili velik posao, ili gotovo nemoguć ukoliko servisi čuvaju stanje (za ispravan rad podaci se moraju čuvati na dulje vrijeme i biti sinkronizirani između upita).

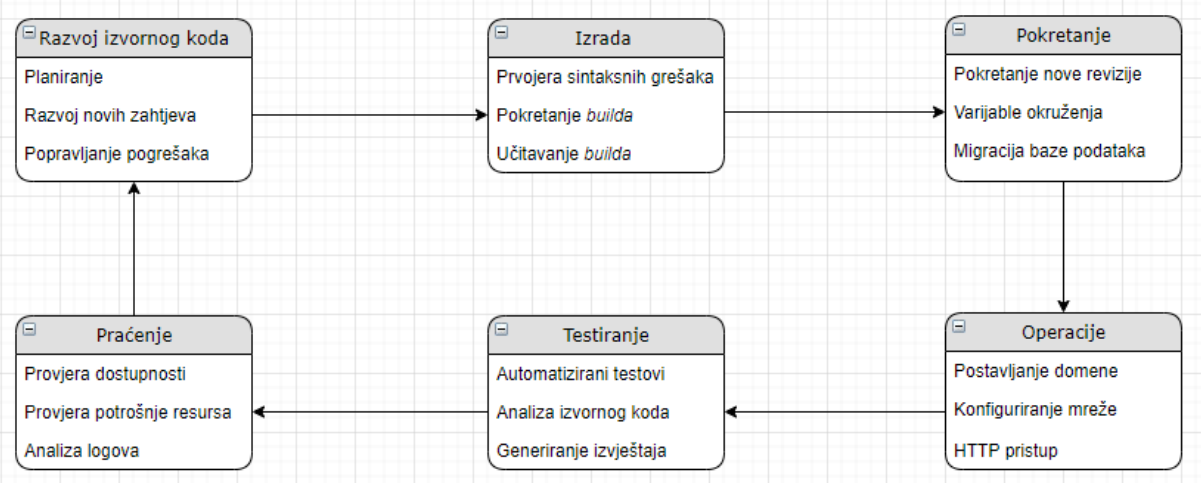
Drugi problem, povezan s međuovisnosti su uska grla. To su najčešće servisi koji čuvaju stanja i jako ih je teško raspodijeliti na više računala, primjerice baze podataka. Ako dođu pod preveliko opterećenje, usporavaju cijeli sustav. Najčešća uska grla su prostor za pohranu datoteka, količina operacija relacijske baze podataka, količina memorije memorijske baze podataka, iskorištenje središnje procesorske jedinice (engl. *Central Processing Unit - CPU*) poslužitelja i slično.

3.3 Opis principa DevOps okruženja

Osnovno je zaduženje kvalitetnog DevOps okruženja pružiti prigodna rješenja na probleme postavljanja programskog koda i skaliranja računalnih resursa kroz jednostavno okruženje za razvoj aplikacije. Suvremeni operacijski dio uvelike je olakšan pogodnostima i servisima koje nude oblaci računala, a većina posla jest integrirati aplikaciju na postojeću arhitekturu.

To se najčešće radi izradom standardnih automatiziranih procesa koji se automatski pokreću kada se dogodi neki događaj. Primjer takvog procesa grafički je prikazan u slici 3.1. Kada se dogodi promjena programskog koda na glavnoj grani:

1. Pokrenuti pretvaranje novog programskog koda u datoteku koja se može izvršavati (engl. *build*).
2. Pokrenuti novu reviziju krajnje aplikacije s novim programskim kodom
3. Toj reviziji predati varijable okruženja koje određuje razvojni tim.
4. Omogućiti toj reviziji pristup bazi podataka i pokrenuti migracijske skripte.
5. Osigurati pristup servisu preko protokola za prijenos hiperteksta (engl. *Hyper Text Transfer Protocol* - HTTP) te mapirati domenu.
6. Pokrenuti automatizirane testove i generirati izvješće.
7. Kontinuirano pratiti ponašanje sustava nakon promjena.



Slika 3.1 Skica jednog primjera automatiziranog procesa

DevOps principi dogovoreni su principi izrade razvojnog okruženja koji za namjenu imaju unaprijediti razvoj aplikacije u vidu sljedećih ciljeva:

- Ušteda vremena i novca – razvojni tim može se fokusirati na izradu potrebne logike
- Brže postavljanje okruženja – kada je potrebno, nova instanca aplikacije (primjerice prezentacijska) jednostavno se i lako doda
- Brži uvod kadra u razvojni tim – ako je sređeno postavljanje lokalnog okruženja, novi članovi neće imati problem postaviti kompleksne aplikacije
- Pouzdanost – izvođenje automatiziranih procesa znatno je pouzdanije od ručnog postavljanja
- Kontinuirana integracija – automatizirana integracija svih faza razvoja digitalnog proizvoda uz minimalno povećanje troška

3.4 Opis izrade DevOps okruženja

Kako bi se na primjeru prikazala izrada ovakvog okruženja, potrebno je izraditi okruženje za *build* aplikacije i način na koji ona dolazi do krajnjeg poslužitelja. Potom, potrebno je osigurati da je taj krajnji poslužitelj skalabilan i da ima infrastrukturu za pohranu podataka koji opstaju između HTTP upita. Konačno, potrebno je osigurati sigurnost pristupa i zaštitu od nekih poznatih napada.

Kako bi se jednostavno moglo skalirati računalne resurse, potrebno je koristiti usluge računarstva u oblaku. Iako bi se u teoriji moglo skalirati do visoke razine i na fizičkim i na virtualnim strojevima, taj bi pristup bio iznimno skup za veliku većinu kompanija. Pružatelji usluga računarstva u oblaku imaju pristup vrhunskoj infrastrukturi, optimiziranoj potrošnji energije, hlađenju u optimalnim uvjetima, računalnoj snazi raspodijeljenoj po cijelom svijetu,

niskoj nabavnoj cijeni i vrhunskim stručnjacima. Doseći kvalitetu računalne infrastrukture koju je moguće iznajmiti je nemoguće ili neisplativo velikoj većini kompanija.

Često se koriste Amazon web usluge (engl. *Amazon Web Services - AWS*), Microsoft Azure ili Google platforma u oblaku (engl. *Google Cloud Platform - GCP*). Za ovaj rad izabrana je GCP zbog dvije osnovne prednosti u odnosu na ostale dvije. Prva je prednost ukupno niža cijena i bolja kontrola budžeta – na ostale dvije platforme puno je lakše greškom doći do velikog računa, dok će GCP prestati raditi te će tražiti od korisnika potvrdu naplaćivanja ukoliko je budžet prekoračen. Druga je prednost takozvano skaliranje do nule, odnosno jedino GCP trenutno nudi mogućnost da aplikacijski servisi drže računalne resurse u hladnoj pričuvi te ne troše budžet dok se ne koriste. Za jednostavnost korištenja, platforma se ne koristi na razini infrastrukture nego na razini platforme.

Za izradu aplikacije za Linux operacijski sustav, najčešće korištena tehnologija je docker, koja se koristi i u ovom eksperimentu, a bit će detaljnije objašnjen u poglavlju 4.1. Za slanje tako izrađene aplikacije na krajnje poslužitelje, često se koriste registri docker slika kao što su Docker hub, a u radu se koristi Container Registry, jer je se dobro integrira s ostalim tehnologijama na platformi. Za skalabilnost krajnjeg poslužitelja popularne su tehnologije poput AWS Lambda ili Funkcije u oblaku, a u ovom radu se koristi Google Cloud Run jer jedina trenutno može pokrenuti docker kontejnere te dolazi i s automatski konfiguriranim zaštitnim postavkama, a bit će dodatno objašnjen u poglavlju 4.2.4. Za pohranu podataka, često se koriste brojne distribucije baza podataka, poput Amazon RDS, a u ovom radu se koristi Cloud SQL, zbog integracije u platformu te će biti dodatno objašnjen u poglavlju 4.2.2.

Za integraciju cijelog okruženja, popularni su alati poput Jenkinsa ili Gitlaba, a u ovom radu koristi se Cloud Build zbog integracije u platformu i zbog automatskog skaliranja te će biti dodatno objašnjen u poglavlju 4.2.4.

4 TEHNOLOGIJE IZVEDBE EKSPERIMENTA

Kako bi postupak izrade kontinuirane integracije i računalne infrastrukture bio što lakši i pouzdaniji, u ovom se radu koriste tehnologije koje će biti opisane u daljnjem tekstu. Za izradu i pokretanje aplikacije koristi se docker, kako bi se omogućilo izvršavanje krajnje aplikacije na pouzdan način neovisno o operacijskom sustavu i infrastrukturi. Ostale tehnologije za računalnu infrastrukturu, automatizaciju procesa i repozitorij programskog koda objedinjene su u Google Cloud Platform te će biti dodatno objašnjene u poglavlju 4.2.

4.1 Docker

Prema opisu u [7], Docker je tehnologija koja služi za razvoj, dostavljanje i pokretanje aplikacija neovisno o računalnoj infrastrukturi na kojoj se aplikacija izvršava. Osnova tehnologije su tzv. docker slike - cjeline koje imaju zaseban linux datotečni sustav, dolaze s potrebnim datotekama za izvođenje nekog programa, kao i sa svim programima o kojima taj program ovisi. Primjerice, aplikacija za obradu videa bi došla u docker slici zajedno s bibliotekom za obradu videa. Izrađena slika može se zatim učitati na registar slika, kojem ostali računalni sustavi mogu pristupiti te se tako olakšava prijenos i razmjena aplikacija između repozitorija programskog koda i krajnjeg poslužitelja.

Međutim, kada se aplikacija pokreće, ona nema samo svoj datotečni sustav, nego postoje i određene odredbe - računalni resursi kojima ima pristup, varijable okruženja, mreža, naredba koja pokreče proces i slično. Kada se pokreće program iz docker slike, definiraju se odredbe, a okruženje u kojem se taj program izvršava naziva se docker kontejner. Svaki ima vlastiti datotečni sustav, koji se izrađuje na osnovu docker slike, s vlastitim odredbama i ne utječu na ostale docker kontejnere.

4.2 Google platforma u oblaku

Platforma koja iznajmljuje infrastrukturu i servise za računarstvo u oblaku. Postoji više razina apstrakcije servisa koje se nude. Moguće je unajmiti virtualne strojeve te ručno upravljati njihovom mrežom, skaliranjem, vatrozidom i slično. Međutim, taj oblik infrastrukture kao servisa (engl. *Infrastructure as a Service - IaaS*) ima određene nedostatke: sklon je pogreškama te je teško naći stručnjake koji mogu kvalitetno složiti vlastitu infrastrukturu. Iz tog je razloga za većinu kompanija skuplji nego da se zakupe gotovi servisi.

S obzirom na navedene nedostatke IaaS oblika, u ovom se radu naglasak stavlja na korištenje servisa, poput platformi na kojima se ne konfigurira poslužitelj (engl. *serverless platform*), baza podataka posluženih na platformi i slično. Ideja je da se platforma sama pobrine o infrastrukturi, mreži i skaliranju, a pruži jednostavno sučelje za upravljanje administratorima. Ovakav pristup poznat je pod nazivom platforma kao servis (engl. *Platform as a Service - PaaS*).

Za potrebe ovoga rada, kao repozitorij programskog koda koristi se Cloud Source Repositories, a za repozitorij docker slika koristi se Container Registry. Za potrebe očuvanja podataka koristi se Cloud SQL, a za konfiguriranje automatiziranih procesa koristi se Cloud Build. Središnja tehnologija koja pokreće krajnju aplikaciju je Google Cloud Run.

4.2.1 Repozitoriji programskog koda i docker slika

Cloud Source Repositories jednostavna je kolekcija udaljenih git repozitorija. Integrira se s ostalim servisima na platformi. Koristi se kao središnji repozitorij programskog koda. Potpuno je bazirana u oblaku i nema potrebe za konfiguriranjem računalnih resursa niti skaliranjem.

Container Registry jednostavna je kolekcija udaljenih docker slika. Nije generalno dostupna i zahtjeva autorizaciju za pristup. Jednostavno se integrira s ostalim servisima na platformi. Svi podaci se potpuno nalaze u oblaku te nije potrebno skalirati resurse ako se broj slika poveća.

4.2.2 Cloud SQL

Servis opisan u [8] koji služi za izradu baze podataka, bez potrebe za održavanjem servera (samoposlužena instanca). Ta je baza prikazana kao servis i nije potrebno voditi računa o poslužiteljima koji pokreću bazu. Servis se sam brine o prostoru za pohranu, repliciranju, sinkronizaciji repliciranja, balansiranju prometa (što je inače iznimno teško izvesti na bazama podataka), izradi sigurnosne kopije i slično.

4.2.3 Cloud Build

Alat na platformi koji omogućava postavljanje samih automatiziranih procesa koji se izvršavaju kada se dogodi određeni događaj (najčešće promjena programskog koda na određenoj grani), tako da se konfiguriraju poslovi (engl. *job*). Iako je u nazivu *build*, moguće

je postaviti više koraka nekog procesa, poput *builda*, postavljanje (engl. *deploy*), testiranje i slično.

Pruža mogućnost izrade okidača (engl. *trigger*), odnosno akcija nakon kojih se neki posao obavlja. To je najčešće nova izmjena programskog koda na Cloud Source Repositories. Svaki se posao sastoji od jednog ili više koraka, a svaki je korak zapravo docker kontejner, koji ne sluša na ulazni promet nego izvrši neku radnju i izađe s izlaznim kodom.

4.2.4 Google Cloud Run

Google Cloud Run pruža mogućnost izvršavanja docker kontejnera u *serverless* okruženju. Za razliku od Cloud Builda, za potrebe ovoga servisa kontejneri ne trebaju izaći s izlaznim kodom i prekinuti izvršavanje, nego se trebaju neprestano izvršavati, odnosno slušati na HTTP promet na određenom *portu*. Cloud Run prosljeđuje tim kontejnerima ulazne HTTP upite i na taj se način vrši komunikacija putem weba.

Cloud Run se sam brine za infrastrukturu, skaliranje računalnih resursa i mrežu. Svaki servis dobije automatski domenu preko koje mu se može pristupiti, a može se i mapirati korisnička domena. Kada dosegne određeni broj simultanih upita prema servisu, servis automatski skalira i dodaje novu instancu servisa. Nije potrebno rukovoditi adresiranjem niti mrežom – kada dolazi upit na upravitelja opterećenja (engl. *load balancer*), on automatski vodi računa o instancama i prosljeđuje upite.

Skaliranje računalnih resursa i konfiguracija mreže obavlja se pomoću znatno veće i poznate tehnologije, Kubernetes. Ona je izrazito kompleksna, ali *Cloud Run* nudi gotovu konfiguraciju Kubernetesa i pruža jednostavno sučelje.

Bitno je za napomenuti nekoliko ograničenja kod aplikacija koje se izvršavaju na Cloud Run platformi opisanih u [9]:

- Kontejneri ne mogu držati stanje izvan jednog HTTP upita, niti u memoriji niti na disku. Trenutno nije moguće očuvati podatke koje se spreme u kontejner. Zbog automatskog skaliranja, nije moguće znati hoće li više upita od jednog klijenta doći na istu instancu, ili hoće li se trenutna instanca nekada ugasiti. Zbog ovoga, podaci se moraju čuvati u vanjskim servisima koji mogu čuvati stanje, i to:
    - datoteke se trebaju čuvati na eksternom disku ili servisu
    - generalni podaci trebaju se čuvati na Cloud SQL ili nekoj drugoj bazi podataka

    - podaci koji trebaju biti pohranjeni u radnoj memoriji, a trebaju biti istovjetni kroz više HTTP upita, trebaju se pohranjivati u memorijsku bazu podataka
- Sav programski kod koji se izvršava mora biti pokrenut preko HTTP upita. HTTP upiti su jedino što Cloud Run prepoznaje kako bi raspodijelio računalne resurse servisu za obavljanje rada.
- Ostala ograničenja, poput maksimalne veličine upita i odgovora, maksimalno vrijeme izvršavanja upita i slično. Cloud Run je zamišljen za HTTP upite i odgovore. Ako je potrebno izvršavanje operacije koja ne može biti izvršena u maksimalnom dopuštenom vremenu, moraju se koristiti drugi servisi.

4.2.5 Cloud CDN i Cloud Armor

Ovi su servisi automatski integrirani u Cloud Run i iznimno su bitni. Djeluju na već konfiguriranim poslužiteljima prije nego što upiti dođu do krajnjih poslužitelja i tako filtriraju i rasterećuju promet krajnje aplikacije.

Sustav za dostavljanje sadržaja (engl. *Content Delivery System* - CDN) osigurava veću brzinu i manju latenciju krajnje aplikacije. Radi na mreži globalno raspodijeljenih računala i izračunava najbolju putanju prometa od klijenta do krajnjeg poslužitelja. Klijent ne komunicira izravno s krajnjim poslužiteljem, nego s poslužiteljem koji je klijentu geografski najbliži. Taj poslužitelj brzom komunikacijom unutar CDN mreže prosljeđuje promet krajnjem poslužitelju. CDN sustav moguće je postaviti da čuva u predmemoriji podatke koji se ne mijenjaju često, poput slika ili skripti, pa često nema potrebe kontaktirati krajnjeg poslužitelja za te resurse. To znatno ubrzava ukupno vrijeme odaziva na HTTP upite klijenta i smanjuje opterećenje krajnjih poslužitelja.

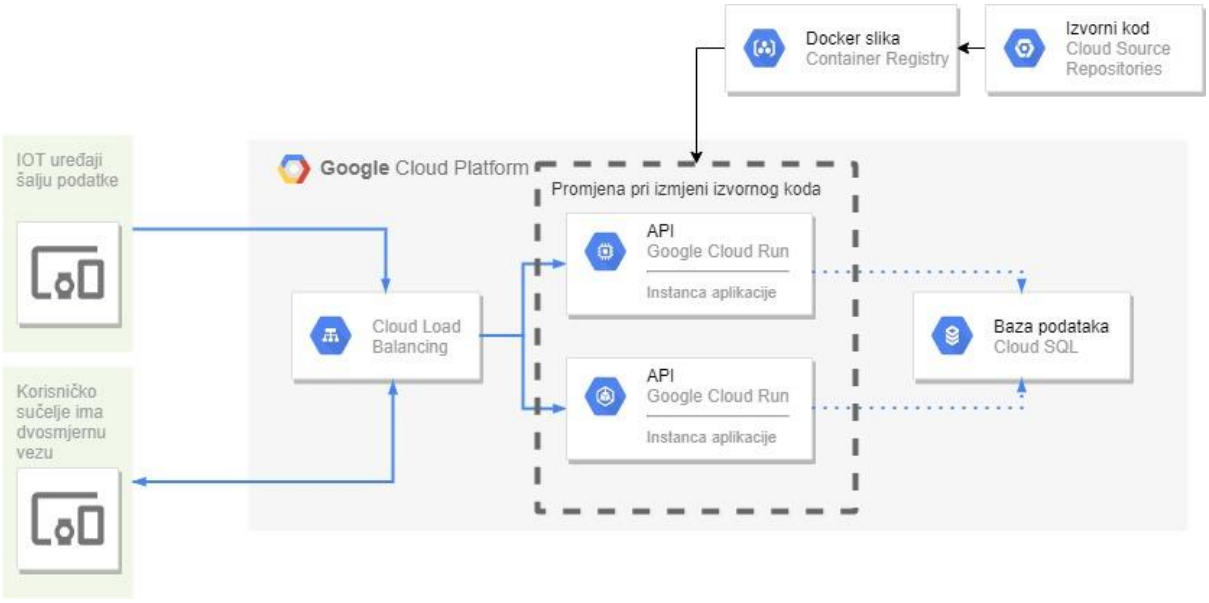
Cloud Armor pruža zaštitu od malicioznih upita i pokušaja napada. Jedan od najvećih problema koje rješava je takozvani napad uskraćivanja usluge (engl. *Denial of Service - DDoS*). Zaštita od DDoS napada zahtjeva veliku računalnu moć, pa je za većinu kompanija iznimno teško ili gotovo nemoguće zaštiti se od DDoS napada vlastitom računalnom infrastrukturom bez pristupa velikoj računalnoj moći koju, primjerice, Google ima. Cloud Armor pokušava blokirati maliciozne upite na prije no što dođu do krajnjih poslužitelja.

5 IZRADA DEVOPS OKRUŽENJA SUSTAVA ZA PRAĆENJE UREDSKOG POSLOVANJA

Kako bi se prikazali principi DevOpsa i utjecaj kvalitetnog DevOps okruženja na olakšan proces izrade aplikacija, u ovom će se poglavlju opisati izrada demonstrativne aplikacije i potrebnog DevOps okruženja za nju.

Glavna je svrha aplikacije upravljanje uredskim poslovanjem. Aplikacija treba moći pružiti sustav odjavljivanja i prijavljivanja korisnika kada ulaze i izlaze iz ureda. Sustav treba zabilježiti vremena prijava i odjava za svakog korisnika zasebno, te prema tim podacima prikazati korisnu analitiku o ukupnom vremenu provedenom u uredu. To treba moći omogućiti za pojedine korisnike ili kumulativno za sve. Zatim, iz tih podataka sustav treba prikazati broj ukupno prijavljenih korisnika u svakom trenutku, te u stvarnom vremenu prikazati obavijesti da se neki zaposlenik prijavio ili odjavio. Konačno, sustav treba prikazivati općenite podatke o temperaturi i vlazi u uredu i ažurirati ih u stvarnom vremenu.

Aplikacija je izrađena prema strukturi klijent-poslužitelj jer različiti klijenti komuniciraju s istim poslužiteljem, što je neophodno u području interneta objekata (engl. *Internet of Things - IOT*) i prikazano je na slici 5.1. IOT uređaji kontaktiraju Aplikacijsko programsko sučelje (engl. *Application Programming Interface - API*) i šalju podatke o temperaturi i vlažnosti. API putem dvosmjerne konekcije javlja korisničkom sučelju na pregledniku trenutna stanja. Svakom zaposleniku pridružuje se identifikator, kojeg uređaj može pročitati i poslati poslužitelju. Na osnovu toga poslužitelj daje osnovne analitike o prisutnosti.



Slika 5.1 Prikaz arhitekture aplikacije u DevOps okruženju

IOT uređaji za mjerenje temperature i vlažnosti zraka, kao i očitavanje korisničkog identifikatora te korisničko sučelje nisu izrađeni u sklopu ovoga rada, no zbog cjelovitosti su prikazane na slici.

5.1 Dizajn baze podataka

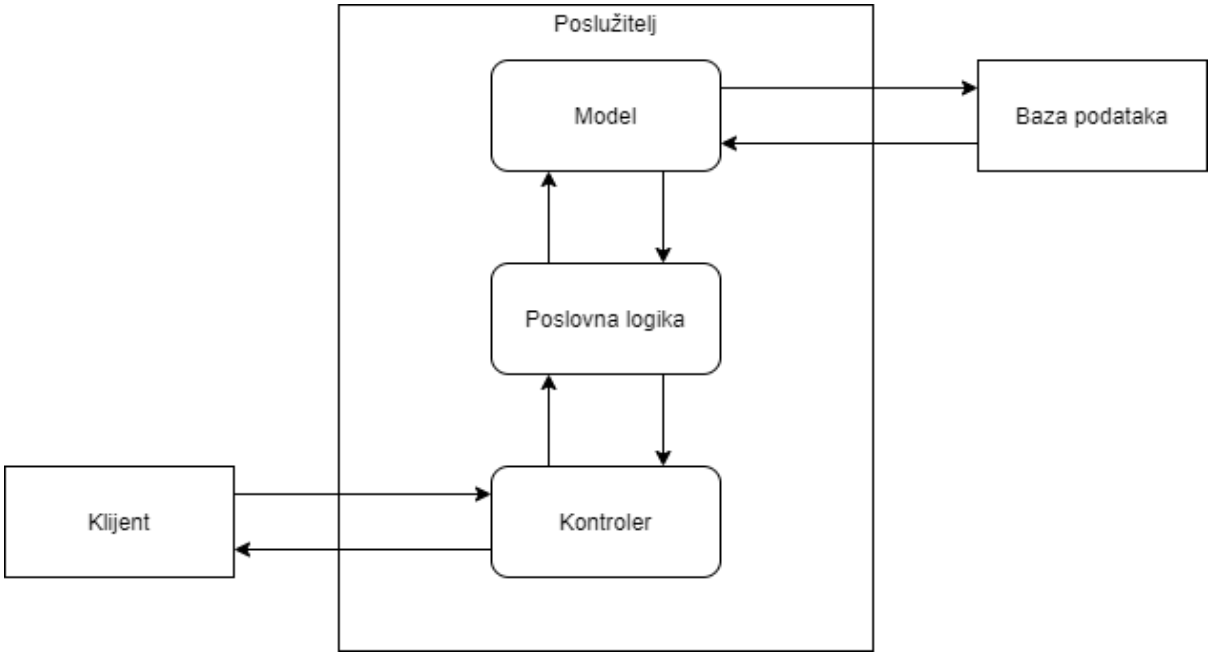
Kako bi podaci ostali očuvani te kako bi ih aplikacija mogla na jednostavan način kasnije dohvatiti i analizirati, koristi se *MySQL* baza podataka.

Glavna je tablica korisnička tablica, koja sadrži osnovne podatke o korisnicima i njihovu razinu pristupa. U njoj se čuvaju podaci o korisnicima poput imena, emaila i identifikatora. Postoji i tablica povijesti pristupa. Ona prati kada se koji korisnik prijavio i odjavio, te na osnovu nje isporučuje analitiku koja se prikazuje na administratorskom panelu. Cijela struktura baze podataka može se vidjeti u migracijskim datotekama u projektu.

Prilikom izrade okruženja ove aplikacije, bitno je imati aktivnu bazu podataka, te se aplikaciji kroz varijable okruženja moraju predati podaci za spajanje na bazu podataka.

5.2 Tehnologije i arhitektura programskog koda aplikacije

Korišteni programski okvir je Adonis 5. To je okvir za pisanje web aplikacija izrađen prema arhitekturi model prikaz kontroler (engl. *Model View Controller - MVC*) koji je skiciran na slici 5.2. Osnova arhitekture su modeli, koji pružaju razinu apstrakcije za komunikaciju s bazom podataka i jednostavan jezik za upite.



Slika 5.2 Prikaz MVC modela uz Client-Server infrastrukturu

Sama tehnologija izrade izvornog koda nije bitna za krajnju infrastrukturu, jer se ona bazira na docker slici, ali bitno je napraviti upute za izradu te slike (napisati *Dockerfile*) za ovu tehnologiju.

5.3 Krajnje točke

Aplikacija s mrežom komunicira preko krajnjih točaka, koje čine API.

Osnovna je krajnja točka za dohvaćanje analitičkih podataka na osnovu prijava i odjava, u kojoj se na dnevnoj bazi računa broj prijavljenih sati, za određenog korisnika ili kumulativno. Isječak iz programskog koda koji je napisan u svrhu analitike prikazan je na slici 5.3.

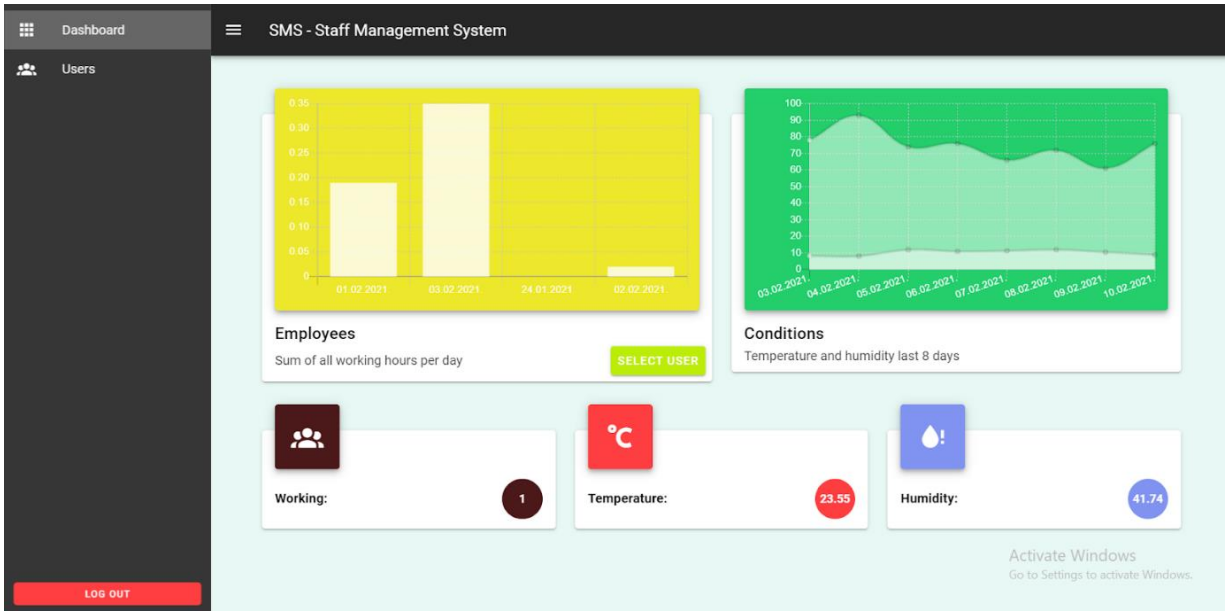
```
public async create (inputs: ExtractRequestInput<AccessAttributeConfig>): Promise<AccessModel> {
  const lastAccess = await AccessModel.query().where(`user_id`, inputs.user_id).orderBy(`created_at`, `desc`).first();
  if (lastAccess) {
    lastAccess.is_active = false;
    await lastAccess.save();
  }

  const type = !lastAccess || lastAccess.type === AccessTypeEnum.left ? AccessTypeEnum.entered : AccessTypeEnum.left;
  // eslint-disable-next-line @typescript-eslint/no-magic-numbers
  const timeFromLastAccess = lastAccess ? Math.abs(lastAccess.created_at.diffNow(`milliseconds`).milliseconds) : 0;
  return await super.create(inputs, {
    type,
    date_of_access: DateTime.local(),
    time_from_last_access: timeFromLastAccess,
    is_active: true,
  });
}

public async getAnalytics (userId?: string): Promise<any> {
  const avgTimeQb = Database.knexQuery().table(AccessModel.table)
    .select(`date_of_access`)
    .sum(`time_from_last_access`)
    .where(`type`, AccessTypeEnum.left)
    .groupBy(`date_of_access`);
  if (userId) {
    void avgTimeQb.where(`user_id`, userId);
  }
  const avgTime = await avgTimeQb;

  const [currentWorking] = await Database.knexQuery().table(AccessModel.table)
    .countDistinct(`user_id`)
    .where(`is_active`, true)
    .where(`type`, AccessTypeEnum.entered);

  return {
    avgTime,
    currentWorking: currentWorking.count,
  };
}
```

Slika 5.3 Prikaz programskog koda za prijavu i analitiku

Zatim, postoje krajnje točke za upise, čitanja, ažuriranja i brisanja (engl. *Create Read Update Delete - CRUD*) korisnika. Preko njih je moguće dodavati korisnike u sustav, ažurirati i micati ih, te najbitnije dodijeliti identifikator.

Također, postoje i krajnje točke za prijavu korisnika. Njih zove IOT uređaj kada očita identifikator, a sustav provjerava ispravnost tog identifikatora te treba li uređaj omogućiti fizički pristup uredu (ako je spojen na bravu). Sustav sam zaključuje radi li se o prijavi ili odjavi, te preko dvosmjerne konekcije javlja administratorima na pregledniku da je došlo do uspješne ili neuspješne prijave.

Konačno, postoji i jednostavna krajnja točka koju uređaj zove periodički, kako bi poslao podatke o temperaturi i vlažnosti zraka. Ti podaci se preko dvosmjerne konekcije šalju web sučelju kako bi se prikazala promjena u stvarnom vremenu. Ukupan prikaz korisničkog sučelja za administratora je prikazan u slici 5.4.



Slika 5.4 Prikaz krajnjeg korisničkog sučelja

5.4 Izrada DevOps okruženja aplikacije

Za ostvarivanje DevOps principa, korištene su tehnologije opisane u poglavlju 4 s Google Cloud Platforme. Postavljen je sljedeći proces:

1. Učitavanje programskog koda na Cloud Source Repositories
2. Izrada *docker* slike
3. Učitavanje slike na Container Registry
4. Postavljanje nove Cloud Run revizije s novoizrađenom slikom
5. Praćenje ponašanja

Ovaj implementirani proces je sličan primjeru danom u poglavlju 3.3, s tim da su određene konkretne tehnologije na kojima se izvršava, dok je u primjeru proces opisan s općenitim pojmovima. Predavanje varijabli okruženja, osiguravanje pristupa bazi podataka i osiguravanje pristupa servisu preko mreže automatski se odrađuju unutar *Cloud Runa*. Za razliku od primjera iz poglavlja 3.3, u ovom implementiranom procesu se ne mapira prava domena i ne pokreću automatizirani testovi. Ovaj proces izvršava Cloud Build. On je konfiguriran u datoteci *cloudbuild.yml*, čiji je krajnji izgled, prikazan u slici 5.5 dosta jednostavan, ali ima mnogo sitnih detalja.

```yaml
steps:
  # Build the container image
  - name: 'gcr.io/cloud-builders/docker'
    entrypoint: /bin/bash
    args:
      - -c
      - |
        (echo "${_GCP_KEYFILE}" | base64 --decode) > gcp-keyfile.json
        docker build -t gcr.io/${_PROJECT_ID}/${_SERVICE_NAME} .

  # Push the container image to Container Registry
  - name: 'gcr.io/cloud-builders/docker'
    entrypoint: /bin/bash
    args:
      - -c
      - |
        docker push gcr.io/${_PROJECT_ID}/${_SERVICE_NAME}

  # Deploy container image to Cloud Run
  - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
    entrypoint: /bin/bash
    args:
      - -c
      - |
        echo "y" | gcloud run deploy ${_SERVICE_NAME} --image gcr.io/${_PROJECT_ID}/${_SERVICE_NAME} --cpu=1
    --region ${_REGION} --timeout=300 ${_MIN_INSTANCES} ${_SQL_CONNECTOR} ${_VPC_CONNECTOR} --memory=256Mi
    --port=8000 --platform managed --set-env-vars $(echo "${_ENV_B64}" | base64 --decode) --update-labels
    service=${_SERVICE_NAME}
        gcloud run services update-traffic ${_SERVICE_NAME} --to-latest --platform=managed --region=${_REGION}
        gcloud beta run services add-iam-policy-binding --region=${_REGION} --platform=managed
    --member=allUsers --role=roles/run.invoker ${_SERVICE_NAME} --no-user-output-enabled
```

Slika 5.5 Prikaz sadržaja cloudbuild.yml datoteke

Kroz datoteku se koriste takozvane varijable zamjene. One služe kako bi se ista logika mogla iskoristiti i za druge projekte, samo s drugim parametrima. Oni se određuju u Cloud Build sučelju, a mnoge su korištene u ovom projektu - primjerice, sadržaj autorizacijske datoteke, ime servisa, identifikator projekta i slično.

5.4.1 Izrada docker slike

Prvi je korak u procesu izrada docker slike, kako bi se osiguralo konzistentno ponašanje na svim računalnim sustavima i njihovim operacijskim sustavima. Budući da se Cloud Build direktno integrira s Cloud Source Repositories, u kontejneru koji se pokreće na Cloud Buildu su automatski dodane datoteke programskog koda.

Kao slika kontejnera izabire se slika koja u sebi ima docker programski paket. Budući da docker ima svoj datotečni sustav, u docker sliku moguće je opet staviti docker programski paket. To služi kako bi bilo moguće izraditi novu sliku unutar kontejnera. Kao naredbu kontejnera odabran je bash terminal s argumentom -c, a treći argument je zapravo lista naredbi izvršavanja.

Sam je proces jednostavan, prvo se izrađuje autorizacijska datoteka (iz varijabli zamjene u lokalni datotečni sustav) za pristup projektu na GCP. Ovaj korak služi kako bi kasnije bilo moguće učitati docker sliku na repozitorij i izraditi novu reviziju:

```
(echo "${_GCP_KEYFILE}" | base64 --decode) > gcp-keyfile.json
```

Potom pokreće se docker alat za izradu slike. Postupak za izradu slike definira se datotekom *Dockerfile*. čiji je izgled prikazan u slici 5.6:

```
FROM node:12-alpine AS base
RUN apk add --no-cache git bash nano
RUN mkdir /home/node/app/ && chown -R node:node /home/node/app
WORKDIR /home/node/app
USER node

FROM base AS dependencies
COPY --chown=node:node package*.json ./
RUN npm set progress=false
RUN npm install

FROM base AS build
COPY --chown=node:node . ./
COPY --from=dependencies /home/node/app/node_modules ./node_modules
RUN node ace build --production

FROM base AS deploy
ENV NODE_ENV=production
COPY --from=build /home/node/app/build ./
RUN touch .env
RUN npm ci --production
CMD node ace migration:run --force && node server.js
```

Slika 5.6 Prikaz datoteke Dockerfile s opisom izrade slike

U prvoj fazi, kopiraju se sve *package.json* i *package-lock.json* datoteke, koje su potrebne kako bi potrebne biblioteke mogle biti instalirane. Ove se dvije datoteke prve kopiraju radi optimizacije korištenja predmemorije – ako prilikom naredbe COPY datoteke nisu promijenjene, docker će znati da nema promjena i da može nastaviti s podacima iz predmemorije, odnosno neće morati svaki puta instalirati biblioteke.

U drugoj se fazi kopiraju ostale datoteke s diska (programski kod). Zatim se pokreće naredbe *build,* koja dolazi s Adonis okvirom te se izvorni programski kod prebacuje u programski kod koji će se izvršavati.

U trećoj se fazi ponovo instaliraju potrebne biblioteke, ali samo one koje su potrebne za produkcijsku upotrebu. Ovo je u izdvojenoj fazi radi smanjenja veličine docker slike, odnosno da biblioteke koje su potrebne u fazi izrade ne budu u fazi izvršavanja.

Konačno, određuje se naredba koja je glavni proces kontejnera. Ona služi samo kao predefinirana naredba, ne izvršava se u fazi izrade slike nego tek kada se kontejner pokreće. Ona se može promijeniti prilikom pokretanja kontejnera. U ovom slučaju, kao naredba se uzima spoj dvije naredbe: jedna je migracija baze podataka (završava s izlaznim kodom), a

druga je pokretanje poslužiteljskog programskog koda (ne završava nego sluša na ulazni promet).

Tako izrađena docker slika se označava oznakom posljednja (engl. *latest*) i diže se na Container Registry jednostavnom naredbom.

5.4.2 Postavljanje slike na Cloud Run

U ovom koraku, docker slika koja se koristi ima u sebi instaliran set alata za izradu softvera (engl. *Software Development Kit - SDK*) za naredbeni redak koja pruža tekstualno sučelje za upravljanje resursima na platformi. Iznimno je korisna za izvođenje automatiziranih procesa.

U ovom se koraku prvo postavlja Cloud Run servis. Izvršava se preko *deploy* naredbe, a prema imenu platforma sama odlučuje treba li izraditi novi servisi ili samo napraviti novu reviziju već postojećeg. U ovoj je naredbi najbitnije odrediti sliku iz koje se izrađuje kontejner – određuje se slika s oznakom *latest* koja je već učitana na Container Registry.

Uz to, određuju se i dodatne odredbe kao količina memorije, virtualnih procesora i varijable okruženja – sve preko varijabli zamjene. U varijablama okruženja moraju se odrediti varijable koje aplikacija zahtijeva, a to su najčešće osjetljivi podaci, podaci za pristup bazi podataka i slično:

```
gcloud run deploy ${_SERVICE_NAME} --image
gcr.io/${_PROJECT_ID}/${_SERVICE_NAME} --cpu=1  } --memory=256Mi --
port=8000 --platform managed --set-env-vars $(echo "${_ENV_B64}" | base64 -
-decode) --update-labels service=${_SERVICE_NAME}
```

Potom, potrebno je preusmjeriti promet sa stare na novu reviziju. Naime, na Cloud Run platformi, moguće je imati više različitih revizija te odrediti koji postotak prometa ide na koju reviziju. To je korisno u nekim slučajevima, no u većini slučajeva sav promet treba biti na posljednjoj reviziji:

```
gcloud run services update-traffic ${_SERVICE_NAME} --to-latest
```

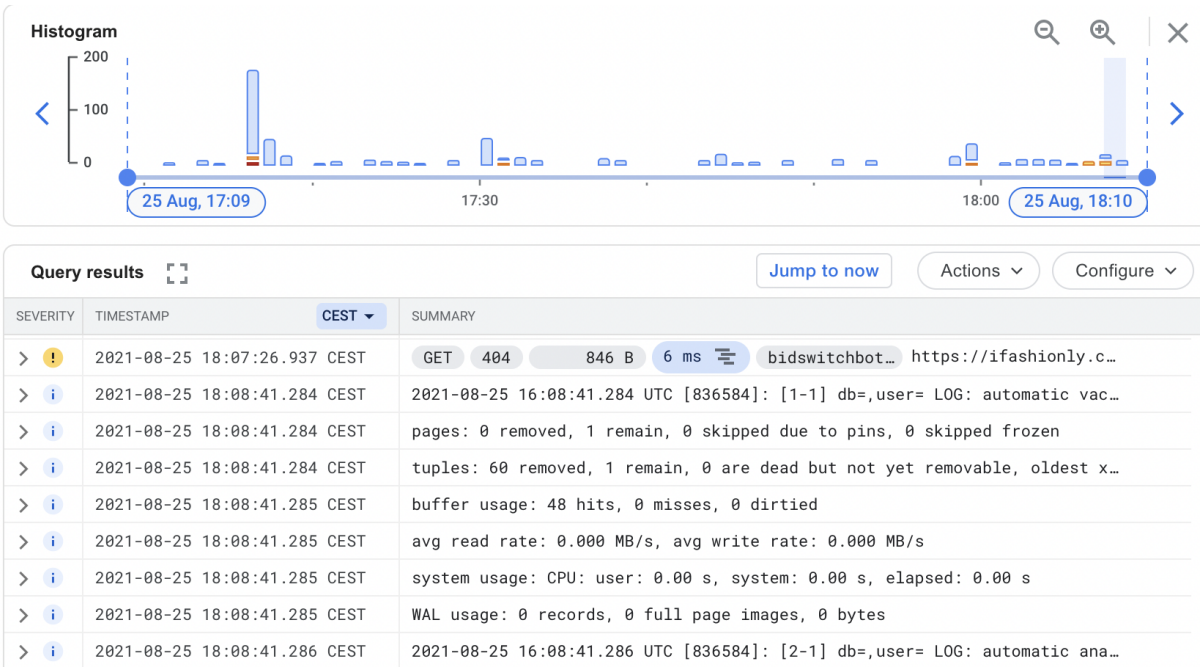
Posljednje, potrebno je osigurati da korisnici mogu pristupiti servisu. Ako nije određeno, pristup će imati samo autorizirani korisnici i servisi. To se radi tako da se svim korisnicima (*allUsers*) pridoda uloga *cloudRunInvoker*:

```
gcloud beta run services add-iam-policy-binding --member=allUsers --
role=roles/run.invoker ${_SERVICE_NAME}
```

5.4.3 Praćenje sustava

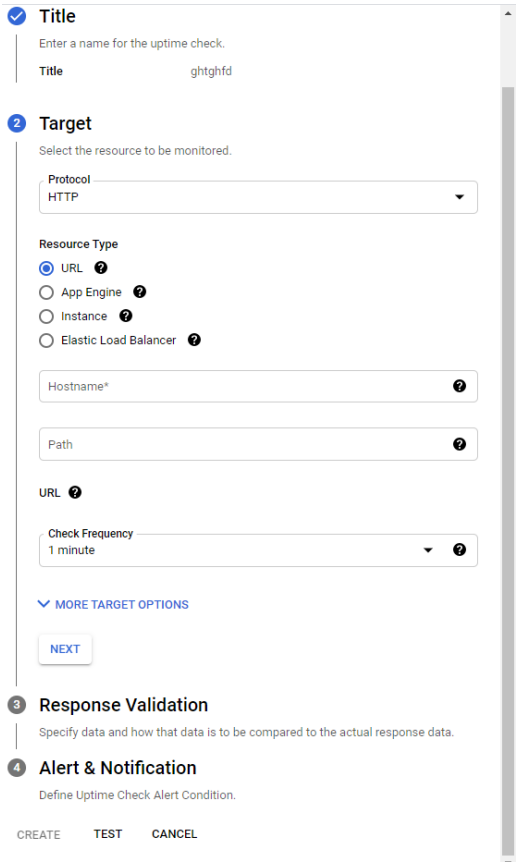
Dodatna mogućnost DevOps okruženja jest kontinuirano praćenje sustava. To uključuje detekciju nedostupnosti sustava, prijavu grešaka, analitiku potrošnje resursa i slično. Na GCP postoji velik broj usluga koje pomažu u ovom koraku.

Za praćenje pogrešaka i logova može se koristiti sustav za praćenje aplikacijskih *logova*, prikazan na slici 5.7. Pruža mogućnosti pregledavanja i filtriranja *logova* iz više različitih servisa.



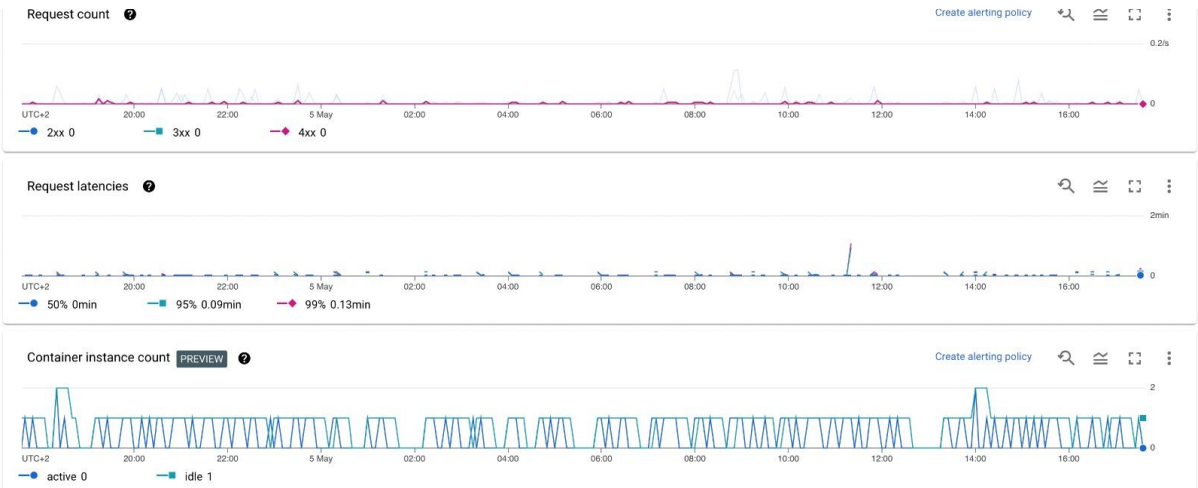
Slika 5.7 Prikaz sučelja za pregled logova

Detekcija nedostupnosti sustava također se može postaviti za korištene servise i postaviti obavijesti kako bi se što prije otklonio potencijalni problem, kao što je prikazano na slici 5.8. Može se postaviti viša ili niža učestalost provjere, kao i dodatni parametri (primjerice provjera HTTP odgovora).



Slika 5.8 Prikaz sučelja za postavljanje provjere dostupnosti

Kao što je prikazano na slici 5.9, moguće je i pratiti količinu resursa koje servis troši, broj instanci, latentnost upita i slično.



Slika 5.9 Prikaz sučelja za praćenje broja instanci i latentnost upita

Na platformi su dostupni i brojni drugi sustavi za praćenje, poput praćenje cijene korištenih usluga, izrada *snapshota* u programskom kodu za lakše otklanjanje greški, pregled sporih upita na bazu podataka, opterećenje baze podataka, postavljanje upozorenja i slično. Korištenjem

platforme može se bez puno truda razviti moćan sustav praćenja za brži pronalazak i uklanjanje grešaka.

5.4.4 Mogućnosti unaprjeđenja DevOps okruženja

Kada se radi bilo kakav ozbiljniji projekt razvoja aplikacije, potrebno je napraviti i razvojno okruženje za razvojni tim i testno okruženje za tim osiguranja kvalitete. Za demonstrativnu aplikaciju ovo naravno nije potrebno, ali inače bi se u ovisnosti o *git* grani ovaj proces izvršavao na različitim instancama s različitim varijablama okruženja.

Jedna faza koja je inače korisna u procesu postavljanja programskog koda je pokretanje automatiziranih testova. Mogli bi se snimiti ili napisati automatizirani testovi te bi se mogli pokrenuti nakon postavljanja novog programskog koda na testnoj ili prezentacijskoj instanci. Uz to, moglo bi se i postaviti analiziranje programskog koda, skeniranje propusta i slično. U ovisnosti o rezultatima, prekinuo bi se proces postavljanja novog programskog koda na produkcijsku instancu.

Konačno, moglo bi se i automatizirati postavljanje cijelog projekta na GCP. Trenutno nije problem ručno postavljanje baze podataka, no kada bi sustav postajao sve kompliciraniji, sve bi teže bilo ručno postaviti projekt. Primjerice, za svako okruženje moralo bi se postaviti relacijska, nerelacijska i memorijska baza podataka, upravitelj reda zadataka, upravitelj vremenski pogonjenih zadataka i slično. Ovo može postati još veći problem ukoliko je sustav koji se izrađuje distribuiran po licencama. U tom slučaju, često bi se morala podizati cijela infrastruktura za veće klijente.

6 ZAKLJUČAK

Sve većim porastom popularnosti suvremenih metodologija izrade programske podrške, raste potreba za standardnim procesima za postavljanje, pregled i praćenje programskog koda. Porastom količine podataka i kompleksnijim zahtjevima aplikacija rastu i zahtjevi za računalnim resursima. Visoka dostupnost aplikacijskih rješenja nužna je za uspjeh aplikacije. Suvremeno tržište ima visoke standarde sigurnosti, kvalitete i dostupnosti programskih rješenja.

Kako bi se ti visoki standardi postigli na brz i učinkovit način, sve su popularnije usluge računarstva u oblaku, i to konkretno PaaS oblik najma infrastrukture. Malen je broj tvrtki koje imaju pristup dovoljnim ljudskim i računalnim resursima da bi im se isplatilo samostalno izraditi arhitekture koje mogu podnijeti navedene zahtjeve.

U radu je opisana jedna platforma za usluge računarstva u oblaku od kompanije Google i opisane su tehnologije i servisi koje platforma pruža za jednostavno integriranje aplikacija modernom pristupu poput Cloud Builda, Cloud Runa i Cloud SQL. Opisani su DevOps principi i ciljevi ispravnog postavljanja DevOps okruženja.

Kako bi se na praktičnom primjeru prikazalo korištenje servisa i usluga računarstva u oblaku za izradu DevOps okruženja, izrađena je demonstrativna aplikacija za praćenje uredskog poslovanja. Na njoj je prikazan proces automatske izrade i podizanja programskog koda, te je prikazano kako se jednostavnim sučeljem može doći do jako skalabilnog produkta, koji se lagano ažurira, kontinuirano integrira i može se efektivno pratiti.

Demonstrirana je važnost izrade ispravnog DevOps okruženja u procesu razvoja aplikacije. Sljedeći DevOps principe prilikom izrade aplikacije i koristeći tehnologije i servise oblaka računala, prikazano je kako je s relativno malo utrošenog vremena moguće doći do pouzdanog sustava koji može skalirati s porastom opterećenja. Konačno, prikazano je i kako ispravno DevOps okruženje može uštediti veliku količinu vremena i smanjiti broj grešaka prilikom postavljanja novog programskog koda i praćenja sustava u produkcijskom okruženju.

LITERATURA

[1] Auto DevOps [online], Gitlab, 2021, dostupno na: https://docs.gitlab.com/ee/topics/autodevops?search=autodevops [12.5.2021.]

[2] D. Sugden, Get started with CI/CD using github actions [online], Medium, 2020, dostupno na: https://medium.com/swlh/get-started-with-ci-cd-using-github-actions-ca32d34b2943?session_id=hgsdkfkujdfsabidfaba [14.5.2021.]

[3] M. Balliauw, Cloud Integrations [online], JetBrains, 2013, dostupno na: https://www.jetbrains.com/teamcity/features/cloud_integrations.html [1.6.2021.]

[4] Package, ship, and re-use config with orbs [online], CircleCI, 2021, dostupno na: https://circleci.com/orbs/?connectedFrom=chrome [1.6.2021.]

[5] App Engine [online], Google, 2018, dostupno na: https://cloud.google.com/appengine?authuser=1&authstrategy=passthrough [2.6.2021.]

[6] What is DevOps [online], Amazon, 2020, dostupno na: https://aws.amazon.com/devops/what-is-devops/?user=0&device=bbaataa [15.5.2021.]

[7] Docker overview [online], Docker Hub, 2013, dostupno na: https://docs.docker.com/get-started/overview/?referal=JsCody [17.5.2021.]

[8] Cloud SQL features [online], Google, 2015, dostupno na: https://cloud.google.com/sql/docs/features? ?authuser=1&authstrategy=passthrough [20.5.2021.]

[9] Quotas and Limits [online], Google, 2021, dostupno na: https://cloud.google.com/run/quotas?authuser=1&authstrategy=passthrough [21.5.2021.]

SAŽETAK

Ispravno DevOps okruženje vrlo je važno za svaki ozbiljan razvoj programskog rješenja. Štedi vrijeme razvojnom timu, smanjuje mogućnost grešaka i daje pouzdanu kontinuiranu integraciju programskog koda u suvremenom razvojno okruženje.

U radu je detaljno opisana Google Cloud Platform i brojni servisi na njoj koji pružaju pristup kompleksnim platformama kroz jednostavno sučelje. Prikazano je korištenje PaaS oblika računarstva u oblaku radi smanjenja vremena utrošenog na konfiguraciju infrastrukture. Kako bi se prikazalo ispravno korištenje servisa izrađena je demonstrativna aplikacija za praćenje uredskog poslovanja. Za tu je aplikaciju izrađeno DevOps okruženje koje osigurava pouzdano automatsko postavljanje novog programskog koda, skalabilnost računalne infrastrukture i način praćenja ponašanja aplikacije u produkcijskom okruženju.

Ključne riječi: računarstvo u oblaku, DevOps, Google cloud platforma, kontinuirana integracija

ABSTRACT

DEVOPS ENVIRONMENT ON GOOGLE CLOUD PLATFORM

The correct DevOps environment is extremely important part of any serious software development. It saves time for development team, lowers rate of errors and offers reliable continuous integration of code in modern environment.

Google Cloud Platform has been explained in detail, with numerous services that run on top it and provide access to complex platforms through simple interface. PaaS form of cloud computing has been demonstrated to reduce time spent on infrastructure configuration. A demonstrative application for office management was built to showcase the correct service usage. On top of the application, a DevOps environment was implemented to ensure reliable and automated code deployment, infrastructure scalability and the system for monitoring application behavior in production.

Keywords: cloud computing, DevOps, Google cloud platform, continuous integration

ŽIVOTOPIS

Ivan Jakab, rođen 26. lipnja 1997. u Vinkovcima. Osnovnoškolsko obrazovanje završio u „OŠ Ivana Brlić-Mažuranić Rokovci-Andrijaševci“. Pohađao matematičku gimnaziju Matije Antuna Reljkovića u Vinkovcima i maturirao 2016. godine. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija na sveučilištu Josipa Jurja Strossmayera u Osijeku. Stječe status sveučilišnog prvostupnika računarstva 2019. godine. Upoznat s procesom izrade web aplikacija, te trenutno zaposlen u Gauss developmentu.