DevOps metodologija razvoja programskih rješenja

Krajinović, Marko

VELEUČILIŠTE U RIJECI

Marko Krajinović

DEVOPS METODOLOGIJA RAZVOJA PROGRAMSKIH RJEŠENJA

(završni rad)

Rijeka, 2021.

VELEUČILIŠTE U RIJECI

Marko Krajinović

DEVOPS METODOLOGIJA RAZVOJA PROGRAMSKIH RJEŠENJA

(završni rad)

MENTOR

dr. sc. Ida Panev, viši predavač

STUDENT

Marko Krajinović

MBS: 2422000024/15

Rijeka, rujan 2021.

VELEUČILIŠTE U RIJECI

Poslovni odjel

Rijeka, 22. ožujka 2021.

ZADATAK
za završni rad

Pristupniku MARKU KRAJINOVIĆU **MBS: 2422000024/15**

Studentu preddiplomskog stručnog studija Informatika izdaje se zadatak za završni rad – tema završnog rada pod nazivom:

DEVOPS METODOLOGIJA RAZVOJA PROGRAMSKIH RJEŠENJA

Sadržaj zadatka:
Objasniti DevOps metodologiju, usporediti s ostalim srodnim metodologijama, opisati ciljeve DevOps-a i alate koji se koriste. Praktično pokazati metodu stalne integracije i isporuke (Continuous Integration / Delivery).

Preporuka:
Rad obraditi sukladno odredbama Pravilnika o završnom radu Veleučilišta u Rijeci.

Zadano: 22. ožujka 2021. **Predati do: 15. rujna 2021.**

Mentor:
Dr. sc. Ida Panev, viši predavač

Pročelnik odjela:
Dr. sc. Anita Stilin, viši predavač

Zadatak primio dana: 22. ožujka 2021.
Marko Krajinović

Dostavlja se:
- **mentoru**
- **pristupniku**

Prilog 5.

I Z J A V A

Izjavljujem da sam završni rad pod naslovom DevOps metodologija razvoja programskih rješenja izradio samostalno pod nadzorom i uz stručnu pomoć mentora Dr. sc. Ide Panev.

Marko Krajinović

Krajinović

(potpis studenta)

Sažetak rada

DevOps metodologija je skup praksi koji kombinira razvoj programskog rješenja i IT operacije (eng. development i eng. operations). Jedna je od najnovijih metodologija kojima se razvojni timovi koriste prilikom razvoja programskih rješenja. Počeci DevOps metodologije bili su 2008. godine, a tek u posljednjih nekoliko godina stiče veću popularnost unutar IT zajednice. Sve veću popularnost stiče zbog jednostavnosti korištenja, korištenja najmodernijih tehnologija i alata te brzine razvoja programskog rješenja na ovaj način. DevOps metodologija popularna je kako među razvojnim timovima, tako i među krajnjim korisnicima programskih rješenja, prvenstveno zbog uštede vremena ali i potrebnih resursa za razvoj i korištenje programskih rješenja.

Ključne riječi: DevOps, Waterfall, Agile, Scrum, DevSecOps,

Sadržaj

1. Uvod

U proteklih nekoliko godina unutar IT zajednice sve je popularnija DevOps metodologija razvoja programskih rješenja. Razvojnim timovima olakšava i ubrzava razvoj programskih rješenja, na zadovoljstvo razvojnih timova i krajnjih korisnika. U ovom završnom radu opisati će se DevOps metodologiju razvoja programskih rješenja i njezinu primjenu u sustavu stalne integracije i isporuke programskog rješenja. Usporediti će se DevOps metodologiju s ostalim srodnim metodologijama (Waterfall, Agile, DevSecOps), te objasniti njene prednosti i nedostatke u odnosu na ostale srodne metodologije. Opisati će se ciljevi DevOpsa i neki od alata koji se koriste prilikom razvoja programskog rješenja DevOps metodologijom. Na primjeru jednostavne Web aplikacije napisane u Node.js programskom jeziku, prikazati će se postavljanje i način korištenja *Continuous integration* (CI) / *Continuous delivery* (CD) metode uz korištenje distribuiranog sustava za upravljanje izvornim kôdom GitHub te njegove značajke Actions.

2. Što je DevOps?

DevOps je skup praksi koji kombinira razvoj programskog rješenja i IT operacije (eng. development i eng. operations). Ova metodologija, iako je svoje početke doživjela 2008. godine, tek u posljednjih nekoliko godina postiže veću popularnost unutar IT zajednice.

Idejni tvorac DevOpsa je Patrick Debois. Na ideju DevOpsa došao je radeći kao tester na projektu 2007. godine. Na Agile Infrastructure konferenciji 2008. godine upoznaje Andrewa Shafera s kojim diskusira o svojim idejama. Veće zanimanje za DevOps unutar krugova IT zajednice počelo je nakon prezentacije „10+ Deploys a Day: Dev and Ops Cooperation at Flickr" 2009. godine. Iste godine održana je i prva konferencija DevopsDays u Belgiji. Od 2010. godine ova konferencija se održava u mnogo različitih gradova diljem svijeta. Alanna Brown 2012. godine objavljuje prvi State of DevOps Report, koji se od 2014. objavljuje svake godine. Knjiga The Phoenix Project, autora Genea Kima, Kevina Behra i Georgea Spafforda, objavljena je 2013. godine te je stekla veliku popularnost, te doprinjela upoznavanju šire publike s pojmom DevOpsa. Tvrtka Forester Research proglasila je 2017. godinu za „Godinu DevOpsa" te objavila kako prema njihovim istraživanjima gotovo 50% tvrtki implementira DevOps. Od onda do danas sve više tvrtki i razvojnih timova implementira i koristi DevOps metodologiju u razvoju programskih rješenja. [1]

U tradicionalnom timu za razvoj programskih rješenja, najčešće su osoba koja razvija programsko rješenje i osoba zadužena za njegovu isporuku i implementaciju dvije različite osobe. Pojavom novih načina rada i tehnologija, kao što su npr. metode stalne integracije i isporuke, omogućeno je da oba navedena posla odrađuje jedna osoba koristeći određene alate i tehnologije. Osoba koja radi na ovoj poziciji često ima titulu DevOps inženjera.

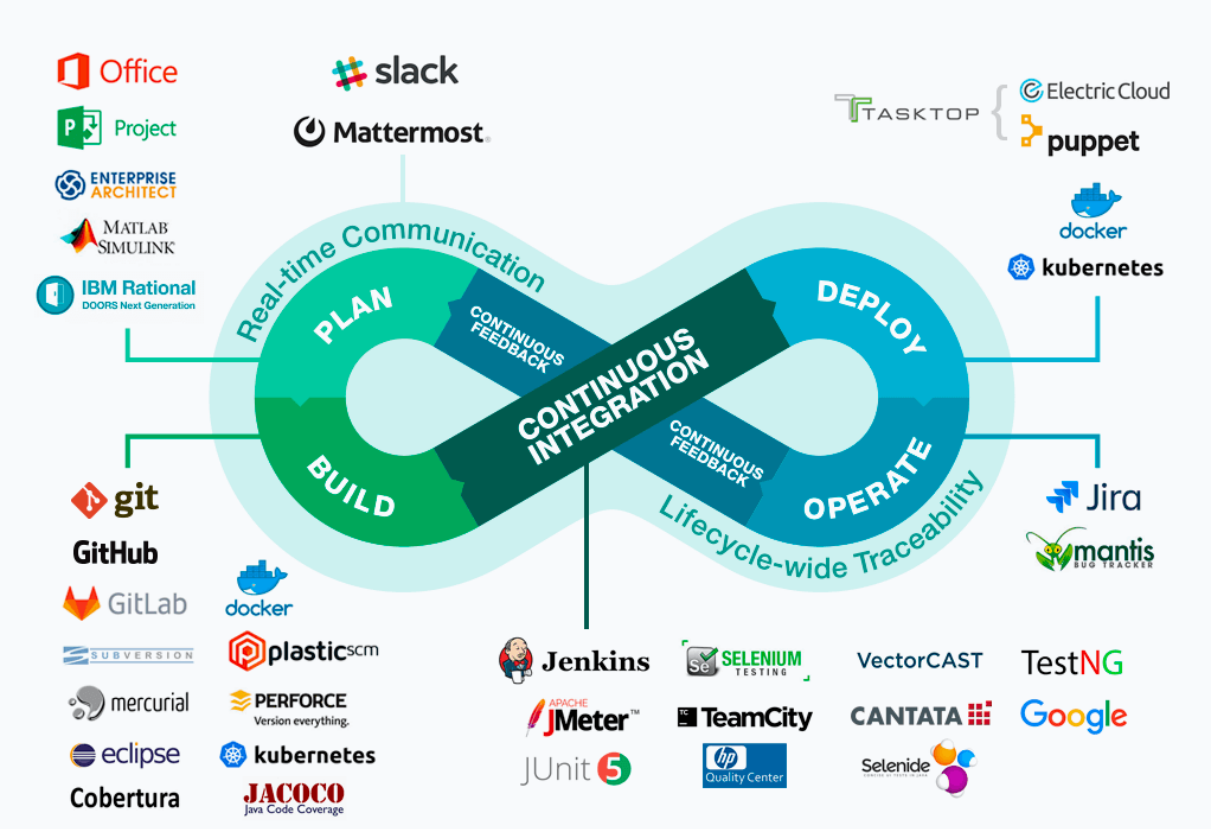
Zadaci DevOps inženjera su, osim razvoja novih funkcionalnosti programskog rješenja ili ispravaka nedostataka u već postojećima, i oni koje bi inače radila osoba u timu zadužena za operacijski dio sustava. Tako su DevOps inženjeri zaduženi za postavljanje i održavanje sustava i alata, kao i samih programskih rješenja. Osim dobrog poznavanja cijelog sustava i programskog rješenja, DevOps inženjer mora i dobro poznavati sustave na kojima se programsko rješenje izvodi, npr. Linux operativni sustav na lokalnom poslužitelju ili u oblaku, baze podataka, alate za upravljanje korisničkim dozvolama i pravima na poslužitelju, web poslužitelje itd.

DevOps inženjeri, osim dobrog poznavanja alata koje koriste za razvoj programskih rješenja, moraju dobro poznavati i DevOps alate. Tako na primjer moraju biti dobro upoznati s distribuiranim sustavima za upravljanje izvornim kôdom (GitHub, Mercurial), sustavima računarstva u oblaku (AWS, Azure, Google Cloud), sustavima za virtualizaciju (Docker) te sustavima za klasterizaciju (Kubernetes), alatima za automatizaciju infrastrukture (Chef, Puppet, Ansible), alatima za korištenje metode kontinuirane integracije i kontinuirane isporuke (Jenkins, Travis CI) i alate za promatranje performansi programskih rješenja (Datadog, Newrelic). Slika 1 prikazuje zadatke DevOps inženjera i neke od alata koji se koriste.

Slika 1 prikazuje procese koji se odvijaju prilikom razvoja programskog rješenja u razvojnim timovima koji rade s DevOps metodologijama razvoja. Procesi prikazani na slici su: planiranje programskog rješenja, njegova izgradnja, kontinuirana integracija (eng. continuous integration), isporuka programskog rješenja, njegova uporaba i kontinuirano primanje povratnih informacija o programskom rješenju od krajnjeg korisnika. Ovi procesi odvijaju se ciklički, odnosno u beskonačnoj petlji kako je to prikazano i na slici. Sve te procese prati i kontinuirana komunikacija s krajnjim korisnikom te praćenje svih aktivnosti tijekom izrade programskog rješenja. Osim procesa koji se odvijaju prilikom razvoja programskog rješenja,

na slici su za svaki od procesa prikazani i neki od popularnih alata koji se koriste u tom procesu.

Slika 1: DevOps aktivnosti i neki od alata koji se koriste



Izvor: https://intland.com/codebeamer/devops-it-operations/ (21.08.2021.)

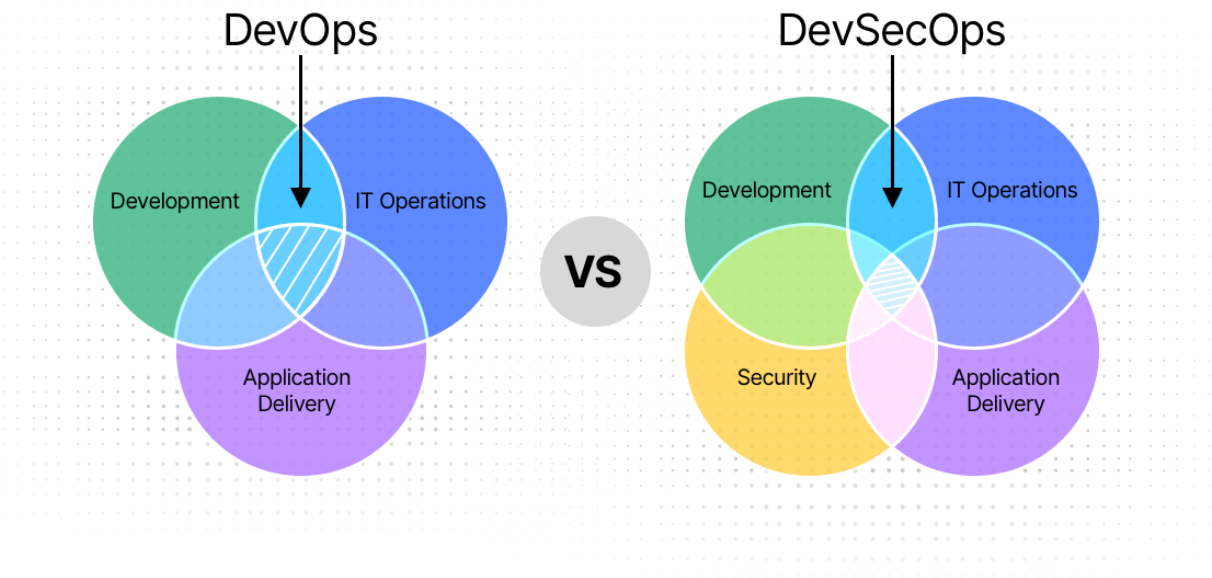
Ciljevi DevOps metodologije razvoja programskih rješenja su integracija razvoja programskih rješenja i IT operacija, kako bi se razvoj programskih rješenja ubrzao, olakšao i pojednostavio, kako razvojnim timovima tako i krajnjim korisnicima. Zadatak DevOps inženjera je automatizacija gotovo svih procesa u životnom ciklusu razvoja programskih rješenja korištenjem modernih alata i sustava. Osim automatizacije, cilj DevOpsa je i pojednostavljenje razvoja programskih rješenja, ubrzanje razvoja programskih rješenja te smanjenje troškova razvoja, izvođenja i uporabe programskih rješenja.

U posljednje vrijeme unutar IT zajednice pojavljuje se i pojam DevSecOps. DevSecOps - razvoj (eng. development), sigurnost (eng. security), operacije (eng. operations). Ovaj pojam

predstavlja prirodnu evoluciju DevOpsa na način da se kroz tipične DevOps procese automatizacije integriraju sigurnosne značajke u programsko rješenje. U prošlosti su sigurnosne značajke vrlo često bile samo dodatak programskom rješenju. U današnjem svijetu informacijskih tehnologija, kada se sve mijenja vrlo brzo te se novi sigurnosni problemi pojavljuju svakodnevno, postoji potreba da sigurnosne značajke budu integralni dio programskog rješenja. DevSecOps integrira sigurnosne značajke u sve Agile i DevOps procese i alate. Bavi se rješavanjem sigurnosnih problema čim se oni pojave, kada ih je lakše, brže i jeftinije ispraviti. Za sigurnost infrastrukture i programskih rješenja u DevSecOpsu zaduženi su svi članovi razvojnog tima, a ne pojedinci. [2]

Slika 2 prikazuje razlike tradicionalnog DevOpsa i DevSecOpsa. Na slici je moguće vidjeti Vennove dijagrame koji pokazuju kako je DevOps skup razvojnih (eng. development), operativnih (eng. operations) i aktivnosti koje se odnose na isporuku programskog rješenja krajnjem korisniku (eng. application delivery), odnosno kako je DevSecOps skup svih prethodno navedenih aktivnosti uz dodane aktivnosti povezane sa sigurnošću programskog rješenja (eng. security) te time objašnjavaju razliku između ova dva pojma.

Slika 2: Razlika DevOpsa i DevSecOpsa



Izvor: https://www.imperva.com/learn/application-security/devsecops-devops-security/ (21.08.2021.)

2.1. DevOps alati - Git

Primarni alat kojeg koriste razvojni timovi koji prakticiraju DevOps metodologiju razvoja programskog rješenja je distribuirani sustav za upravljanje izvornim kôdom. Najpopularniji takav sustav unutar IT zajednice je Git. Git sustav za upravljanje izvornim kôdom nastao je 2005. godine, a njegov kreator je Linus Torvalds. Nastao je kao odgovor na nedostatke do tada postojećih sustava. [3] Sustav Git može se pokretati lokalno, na računalu programera, na lokalnom poslužitelju ili na sustavu u oblaku. Najpopularnija implementacija sustava Git u oblaku je GitHub. Postoje i druge implementacija, kao što su GitLab, Bitbucket, SourceForge i druge.

U radu se za programsko rješenje kreira udaljeni repozitorij na poslužitelju, unutar sustava Git. U udaljeni repozitorij šalje se sav kôd te sve izmjene napravljene na njemu. Sustav Git sprema sve verzije kôda poslane u njega. Napravljena izmjena poslana u udaljeni repozitorij

naziva se *Git commit*. Svaka izmjena poslana na sustav Git mora biti dokumentirana. Dokumentacija poslanih izmjena piše se kao komentar u *Git commitu* te se skupa s izmjenom šalje u udaljeni repozitorij. Sustav za upravljanje izvornim kôdom pamti sve izmjene u programskom

Jedna od značajki gotovo svih distribuiranih sustava za upravljanje izvornim kôdom, pa tako i sustava Git, je grananje (eng. branching). Grananje omogućava spremanje više različitih „kopija“ repozitorija te praćenje svih izmjena kôda poslane u određenu granu (eng. branch). Repozitorij se sastoji od glavnog *brancha (eng. main)*, u kojemu se uobičajeno nalazi provjerena, stabilna verzija kôda. U dodatnim *branchevima* nalaze se verzije kôda na kojima programeri aktivno rade i nad kojima vrše izmjene. Kada se kôd iz dodatnih *brancheva* provjeri, promjene se spajaju (eng. merge) u glavni *branch.*

S udaljenog repozitorija, kôd programskog rješenja mogu preuzeti na lokalno računalo programeri, ukoliko imaju dozvolu za preuzimanje. Osim programera, kôd s udaljenog repozitorija može preuzeti i neki drugi sustav ili alat, ukoliko ima dozvolu za preuzimanje. Ovo omogućava integraciju s drugim sustavima ili alatima, poput sustava računarstva u oblaku ili alata za testiranje programskog kôda, u svrhu *continuous integration / continuous delivery* procesa.

2.2. Sustavi računarstva u oblaku

Kako bi se olakšao razvoj programskog rješenja ali i izvršavanje te korištenje već dovršenog programskog rješenja, koriste se sustavi računarstva u oblaku. Prije pojave sustava računarstva u oblaku, razvojni timovi i korisnici programskih rješenja, svu infrastrukturu i sistemske resurse potrebne za razvoj, izvršavanje te korištenje programskog rješenja morali su osigurati i održavati sami. Ovakav način rada bio je skup i neučinkovit, zahtijevao je kupnju i održavanje infrastrukture i sistemskih resursa te timove sistemskih inženjera koji infrastrukturu i resurse planiraju, grade i održavaju.

Korištenjem sustava računarstva u oblaku, troškovi izgradnje i održavanja infrastrukture i resursa prebacuju se na nekog drugog, što razvojnim timovima i krajnjim korisnicima omogućava uštedu vremena i novca te optimizaciju troškova razvoja i korištenja programskog rješenja.

Prednosti korištenja sustava računarstva u oblaku su mnogobrojne. Glavna prednost korištenja ovakvih sustava je izostanak potrebe za kupnjom i održavanjem resursa, poput poslužitelja ili sustava za spremanje podataka, kao i stručnjaka potrebnih za planiranje i održavanje ovih resursa. Osim toga, sustavi računarstva u oblaku kao prednost imaju veliku dostupnost, pa su tako resursi dostupni od svuda i u svako doba, bez potrebe za postavljanjem dodatne mrežne infrastrukture. Prednost korištenja sustava računarstva u oblaku je i plaćanje točno onoliko resursa koliko je iskorišteno, dok infrastruktura koja nije dio sustava računarstva u oblaku generira troškove bez obzira koristi se ili ne. Još jedna od prednosti je i jednostavno i brzo zakupljivanje dodatnih resursa i infrastrukture na sustavu u računarstvu u oblaku, u nekoliko sekundi i uz minimalne dodatne troškove. Sustavi računarstva u oblaku posjeduju i integrirana sigurnosna rješenja, pa su tako infrastruktura i resursi na sustavima računarstva u oblaku automatski osigurani od potencijalnih sigurnosnih problema.

Osim prednosti, korištenje sustava za računarstvo u oblaku ima i neke potencijalne nedostatke. Upitna je povjerljivost podataka, postoji mogućnost da neovlaštena osoba pristupi podacima kojima ne bi smjela pristupati. Za pristup infrastrukturi i resursima obavezna je konekcija na Internet. Postoji mogućnost nedostupnosti resursa i infrastrukture u slučaju problema sa sustavom računarstva u oblaku, iako su u pravilu ovakvi problemi rijetki i brzo se ispravljaju. Jedan od problema je i „Vendor lock-in“, odnosno otežana migracija resursa i infrastrukture na drugi sustav računarstva u oblaku zbog specifičnosti pojedinih sustava. Iako je prilikom normalnog korištenja cijena resursa i infrastrukture na sustavima računarstva u oblaku u pravilu manja nego korištenje lokalnih resursa i infrastrukture, zbog pogreške korisnika ili neovlaštenog pristupa treće osobe, postoji mogućnost generiranja velikih troškova.[4]

Najpopularniji sustavi računarstva u oblaku su AWS (Amazon Web Services), Microsoft Azure i Google Cloud. U prvom kvartalu 2021. godine AWS je posjedovao 32% tržišta sustava računarstva u oblaku. [5]

2.3. Alati za korištenje metode kontinuirane integracije i kontinuirane isporuke - Jenkins

Alati za korištenje metode kontinuirane integracije i kontinuirane isporuke pomažu programerima, koji rade na razvoju programskih rješenja, automatizirati procese razvoja programskih rješenja, procese testiranja i procese isporuke programskog rješenja krajnjem korisniku.

Jedan od najpoznatijih i najpopularnijih alata za korištenje metode kontinuirane integracije i kontinuirane isporuke je sustav za automatizaciju Jenkins. Ovaj alat dostupan je besplatno, a moguće ga je izvršavati na svim popularnim operacijskim sustavima. Jenkins je programsko rješenje otvorenog kôda te zbog toga ima veliku podršku IT zajednice. Jednostavan je za postavljanje i korištenje, a njegove osnovne mogućnosti moguće je proširiti mnogim dostupnim dodacima. [6]

Jenkins podržava integraciju s većinom drugih distribuiranih sustava za upravljanje izvornim kôdom, integraciju sa sustavima za računarstvo u oblaku kao i ostalim DevOps alatima. Osim Jenkinsa, koriste se i drugi alati, neki od kojih su GitHub Actions, GitLab, AWS CodePipeline, Bamboo i drugi.

2.4. Ostali DevOps alati

Osim prethodno navedenih alata i sustava, DevOps inženjeri često koriste i sustave za virtualizaciju (Docker, Wagrant), sustave za klasterizaciju (Kubernetes, Docker Swarm) te alate za automatizaciju infrastrukture (Chef, Puppet, Ansible) i alate za promatranje performansi programskih rješenja.

Docker sustav za virtualizaciju je alat koji programerima olakšava razvoj programskog rješenja, a krajnjem korisniku njegovu implementaciju te izvršavanje. Docker kontejneri su virtualne mašine koje se pokreću i izvršavaju na računalu domaćinu. Docker kontejneri sadrže operativni sustav i sve ostale pomoćne programe potrebne za izvršavanje programskog rješenja koje se izvršava u Docker kontejneru. Prednost korištenja Dockera i Docker kontejnera je mogućnost definiranja predložaka s uputama koje služe za automatizaciju pokretanja Docker kontejnera i programskih rješenja koja se izvode unutar njih. Također, prednost Docker

kontejnera je i njihova mala veličina, kao i mala potrošnja sistemskih resursa. Na jednom računalu moguće je pokrenuti više Docker kontejnera. Docker kontejnere moguće je pokretati i koristiti lokalno, kao i na udaljenim poslužiteljima odnosno na sustavima za računarstvo u oblaku.

Docker kontejneri mogu se koristiti samostalno ili u skupinama (klasterima). Klasteri su skupine identičnih kontejnera, koji se vrte na istom računalu ili poslužitelju. Svrha klastera je podjela poslova između pojedinih kontejnera, kako bi se poboljšala dostupnost pojedinih kontejnera kao i brzina izvođenja poslova. Svaki od kontejnera u klasteru odrađuje jedan manji dio ukupnog posla, te je tako vrijeme potrebno za odrađivanje ukupnog posla manje. Ukoliko jedan od kontejnera zbog opterećenja ili zbog greške postane nedostupan, poslovi se distribuiraju na ostale dostupne kontejnere. Jedan od najpoznatijih i najpopularnijih alata za izgradnju, održavanje i upravljanje klasterima kontejnera je Kubernetes.

Alati za automatizaciju infrastrukture, neki od kojih su Chef, Puppet i Ansible, omogućavaju automatizaciju postavljanja infrastrukture, upravljanje infrastrukturom i njeno održavanje. Prednost korištenja ovakvih alata je mogućnost pisanja predložaka prema kojima ovi alati postavljaju infrastrukturu, upravljaju njom i održavaju je. Ovakvi alati omogućavaju DevOps inženjerima automatsko postavljanje i pokretanje većeg broja virtualnih poslužitelja ili Docker kontejnera u kratko vrijeme, bez potrebe za ručnim postavljanjem i pokretanjem.

DevOps inženjerima vrlo su bitni i alati za promatranje performansi programskih rješenja (eng. Application performance monitoring tools). Ovi alati u pozadini konstanto promatraju performanse programskih rješenja, te obavještavaju DevOps inženjera o potencijalnim smanjenim performansa. Ovakve probleme se može riješiti ručno, zakupljivanjem dodatne infrastrukture i resursa, a rješavanje ovakvih problema se može i automatizirati. Neki od ovakvih alata su Datadog, NewRelic, AppDynamics i drugi.

Svi najpoznatiji sustavi za računarstvo u oblaku podržavaju korištenje svih prethodno navedenih alata.

3. Životni ciklus razvoja programskog rješenja (SDLC)

Životni ciklus razvoja programskog rješenja je proces koji za rezultat ima programsko rješenje maksimalne kvalitete, uz minimalne troškove, u najkraćem mogućem vremenu. Faze

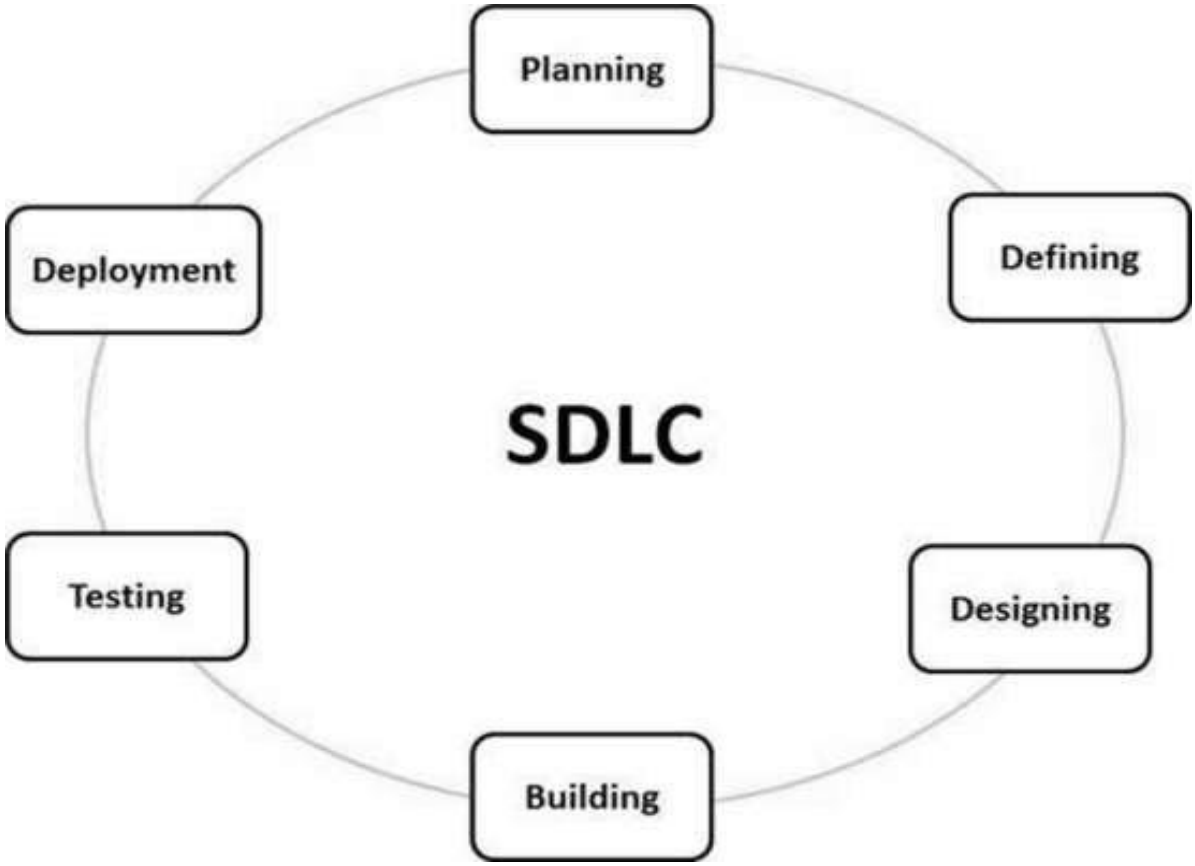
životnog ciklusa razvoja programskog rješenja najčešće su: planiranje i analiza korisničkih zahtjeva, definiranje zahtjeva, dizajn, razvoj rješenja, testiranje rješenja te isporuka i implementacija programskog rješenja. [6]

U prvoj fazi korisnički zahtjevi se analiziraju te se izrađuje plan razvoja programskog rješenja. Ovo je najvažnija faza životnog ciklusa razvoja programskog rješenja. U fazi definiranja korisničkih zahtjeva jasno se određuju i dokumentiraju korisnički zahtjevi iz kojih se u idućoj fazi, fazi dizajna, dizajnira arhitektura programskog rješenja. Faza izgradnje obuhvaća sve aktivnosti na izgradnji programskog rješenja. Nakon faze izgradnje, u idućoj fazi, fazi testiranja, programsko rješenje se testira te se primijećeni nedostaci dokumentiraju i ispravljaju. Posljednja faza je faza isporuke programskog rješenja krajnjem korisniku i implementacije programskog rješenja.

Prvu ideju faznog razvoja programskih rješenja predstavio je 1956. godine Herbert D. Benington. [8]

Slika 3 prikazuje faze životnog ciklusa razvoja programskog rješenja. Na slici je vidljivo kako su faze životnog ciklusa: planiranje (eng. planning), definiranje (eng. defining), dizajn (eng. designing), izrada (eng. building), testiranje (eng. testing) te isporuka programskog rješenja krajnjem korisniku (eng. delivery).

Slika 3: Životni ciklus razvoja programskog rješenja



Izvor: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm (20.08.2021.)

4. Metodologije razvoja programskih rješenja

Postoje različiti pristupi razvoja programskih rješenja koji su opisani u metodologijama razvoja programskih rješenja. Prva nastala metodologija bila je Waterfall, te za nju možemo reći da pripada prvoj generaciji metodologija razvoja programskih rješenja, zajedno s iterativnom, evolucijskom i spiralnom metodologijom. Drugoj generaciji metodologija, koje se i danas koriste, pripadaju Agile metodologije (Scrum, Kanban). Za DevOps možemo reći da je prirodni nastavak Agile metodologija te pripada trećoj generaciji metodologija razvoja programskih rješenja.

5. Waterfall metodologija

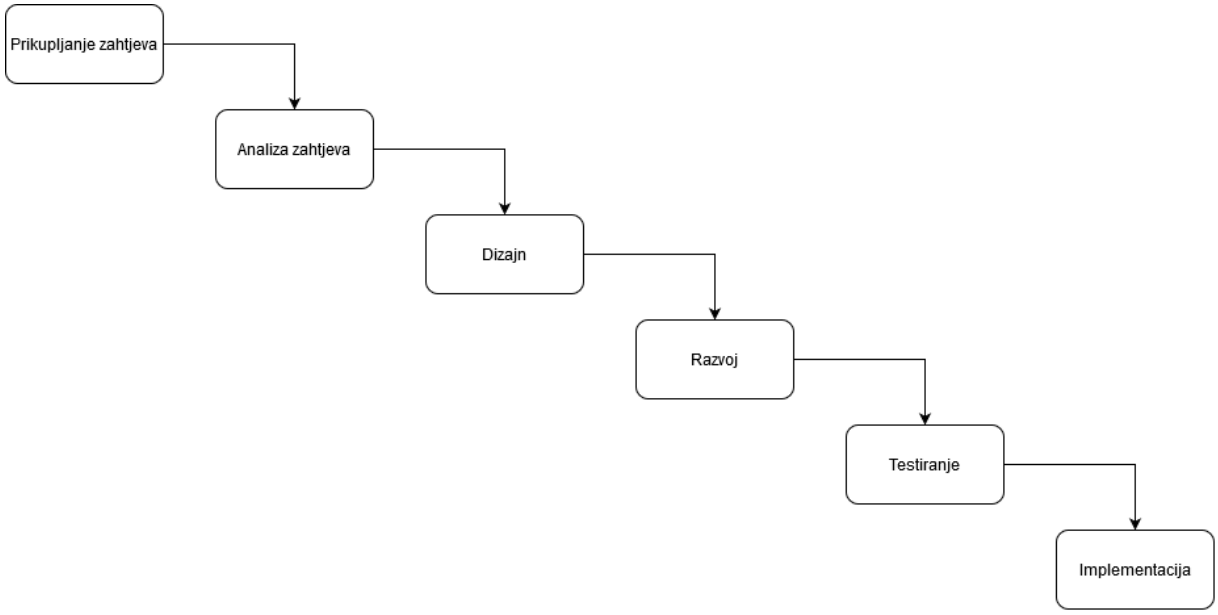
Iako u svom radu nije iskoristio naziv „Waterfall“ ovu metodologiju razvoja programskog rješenja detaljno je opisao 1970. Winston W. Royce. [9]

U Royceovom originalnom opisu navedene su slijedeće faze:

1. Prikupljanje korisničkih zahtjeva
2. Analiza zahtjeva, koja za rezultat ima modele, sheme i poslovnu logiku
3. Dizajn, rezultat kojega je arhitektura programskog rješenja
4. Razvoj programskog rješenja
5. Testiranje
6. Implementacija i održavanje programskog rješenja

Slika 4 prikazuje dijagram prethodno navedenih Waterfall faza razvoja.

Slika 4: Dijagram Waterfall faza razvoja



Izvor: Autor

U Waterfall metodologiji razvoja programskih rješenja, faze razvoja nastupaju točno

određenim redoslijedom te isključivo jedna nakon druge, bez preskakanja faza ili povratka na prethodnu fazu. Waterfall metodologija procesno je orijentirana, što znači da se procesi odvijaju sekvencijalno te nisu fleksibilni. Waterfall metodologija razvoja programskih rješenja u današnje vrijeme je poprilično zastarjela, međutim još uvijek može poslužiti za izradu manjeg i manje kompliciranog programskog rješenja.

6. Agile metodologija

Tradicionalne metode razvoja programskih rješenja kao što je Waterfall nisu prilagođene modernim zahtjevima korisnika, bile su previše spore i ograničene. S vremenom su se pojavile nove metodologije, koje su bile prilagođene novim uvjetima, a neke od kojih su RAD (eng. rapid application development) 1991. godine, Scrum metodologija 1995. godine, XP (eng. extreme programming) 1996. godine.

Godine 2001. u Utahu, sastalo se 17 programera, kako bi raspravljali o tim novim metodologijama. Zaključili su kako metode kojima se koriste imaju neke zajedničke karakteristike. Rezultat njihove rasprave je Manifesto for Agile Software Development, manifest u kojem su objašnjena načela Agile metodologije razvoja programskih rješenja:

„Tražimo bolje načine razvoja softvera razvijajući softver i pomažući drugima pri njegovom razvoju. Takvim radom smo naučili da više cijenimo:

- Ljude i njihove međusobne odnose nego procese i oruđa
- Upotrebljiv softver nego iscrpnu dokumentaciju
- Suradnju s naručiteljem nego pregovaranje oko ugovora
- Reagiranje na promjenu nego ustrajanje na planu.

Drugim riječima, iako cijenimo vrijednosti na desnoj strani, više vjerujemo u one na lijevoj.“ [10]

Osim prethodnog, u Agile Manifestu njegovi autori objavili su i „Načela na kojima se zasniva proglas o agilnom razvoju softvera“. [10]

1. „Rukovodimo se sljedećim načelima: Najvažnije nam je zadovoljstvo naručitelja koje postižemo ranom i neprekinutom isporukom softvera koji nosi vrijednost.

2. Spremno prihvaćamo promjene zahtjeva, čak i u kasnoj fazi razvoja. Agilni procesi uprežu promjene da naručitelju stvore kompetitivnu prednost.
3. Često isporučujemo upotrebljiv softver, u razmacima od nekoliko tjedana do nekoliko mjeseci, nastojeći da razmak bude čim kraći.
4. Poslovni ljudi i razvojni inženjeri moraju svakodnevno zajedno raditi, tijekom cjelokupnog trajanja projekta.
5. Projekte ostvarujemo oslanjajući se na motivirane pojedince. Pružamo im okruženje i podršku koja im je potrebna, i prepuštamo im posao s povjerenjem.
6. Razgovor uživo je najučinkovitiji način prijenosa informacija razvojnom timu i unutar tima.
7. Upotrebljiv softver je osnovno mjerilo napretka. Agilni procesi potiču i podržavaju održivi razvoj. Pokrovitelji, razvojni inženjeri i korisnici trebali bi moći neograničeno dugo zadržati jednak tempo rada.
8. Neprekinuti naglasak na tehničkoj izvrsnosti i dobar dizajn pospješuju agilnost.
9. Jednostavnost – vještina povećanja količine posla kojeg ne treba raditi – je od suštinske važnosti.
10. Najbolje arhitekture, projektne zahtjeve i dizajn, stvaraju samo–organizirajući timovi.
11. Tim u redovitim razmacima razmatra načine da postane učinkovitiji, zatim usklađuje i prilagođava svoje ponašanje.“

7. Usporedba Scrum metodologije i Waterfall metodologije

Scrum je najpopularnija agilna metodologija razvoja programskih rješenja. Nastao je 90-ih godina prošlog stoljeća, a njegovi autori su Jeff Sutherland i Ken Schwaber.

Scrum se sastoji od Scrum timova s njihovim ulogama, Scrum događaja, Scrum artefakata i Scrum pravila. Osnovu Scruma čini Scrum tim, koji se sastoji od stručnjaka kojima je zadatak ostvarivanje ciljeva proizvoda (eng. product). Svaki Scrum tim sadrži vlasnika proizvoda (eng. product owner), razvojni tim (eng. Development team) te *Scrum mastera*.

Vlasnik proizvoda odgovoran je za maksimalizaciju vrijednosti proizvoda koji nastaje kao rezultat rada Scrum tima. [11] Osim toga, vlasnik proizvoda odgovoran je i za upravljanje stavkama u *product backlogu*. *Product backlog* je lista stavki koja se kontinuirano mijenja i nadopunjava. Stavke na listi su poredane po prioritetu izvođenja.

Odgovornost *Scrum mastera* je ispravna implementacija Scruma, ne samo unutar Scrum tima nego i na razini cijele organizacije. *Scrum master* vlasniku proizvoda pomaže upravljati *product backlogom* te ga uči kako kreirati jasne i precizno opisane stavke *product backloga*. Scrum master je odgovoran i za organizaciju Scrum sastanaka po potrebi. Osim toga, odgovoran je i za učinkovitost Scrum tima kojem pomaže u rješavanju problema koji se pojavljuju tijekom rada. [11]

Development tim odnosno članovi *development* tima rade na svim aktivnostima vezanim za izradu inkrementa nakon svakog *Sprinta*. Na početku svakog sprinta zajedno s ostalim članovima Scrum tima dogovaraju na kojim će stavkama iz *product backloga* raditi u tom *Sprintu*. Zadatak *development* tima je na kraju *sprinta* isporučiti uporabljiv te maksimalno kvalitetan inkrement. [11]

Scrum inkrement je proizvod *sprinta* koji je spreman za isporuku krajnjem korisniku. Sastoji se od svih stavki iz svih prethodnih *sprintova*. Krajnji korisnik može odlučiti hoće li ili neće koristiti Scrum inkrement kao programsko rješenje.

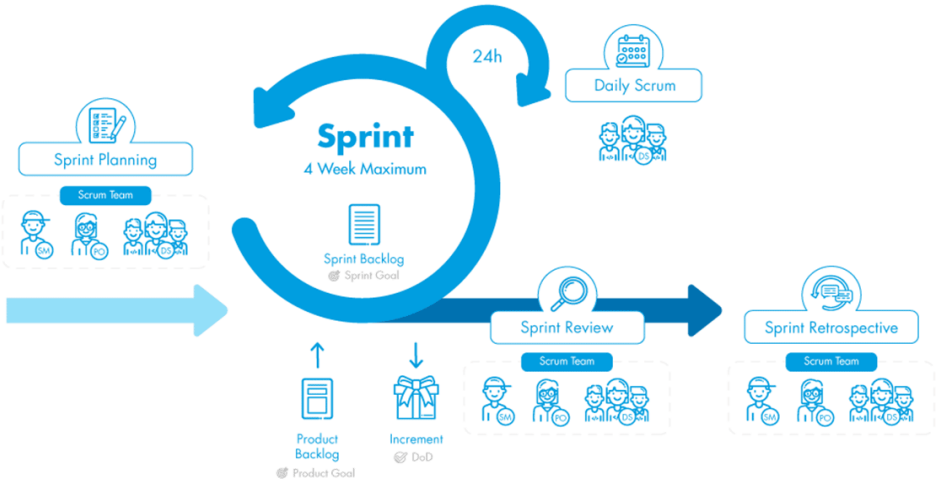
Scrum događaji su unaprijed isplanirana događanja unutar razvoja programskog rješenja kako bi se izbjegla potreba za neplaniranim sastancima prilikom razvoja. Događaji u Scrumu su *Sprint*, Dnevni Scrum sastanci, pregled *Sprinta* (eng. Sprint review) te osvrt na *sprint* (eng. Sprint retrospective). Artefakti u Scrumu, prema Vodiču za Scrum su *Product backlog*, *Sprint backlog* te inkrement. [11]

Sprint je vremensko razdoblje unaprijed dogovorene fiksne duljine. Za vrijeme trajanja *Sprinta* cijeli Scrum tim provodi aktivnosti cilj kojih je isporuka inkrementa. *Product owner* one stavke na kojima će *development* tim raditi za vrijeme trajanja *sprinta* iz *product backloga* prebacuje u *sprint backlog. Development* tim na dnevnim *Daily Scrum* sastancima kreira plan rada za taj dan. Članovi *development* tima jedni druge obavještavaju o stavkama *sprint backloga* na kojima su radili dan ranije, stavkama *sprint backloga* na kojima planiraju raditi na dan sastanka te o zaprekama koje ih sprječavaju u ostvarenju cilja *sprinta.* Nakon završetka *sprinta*

održava se *Sprint review* na kojemu Scrum tim zajedno s krajnjim korisnikom razgovara o izvršenim aktivnostima u proteklom *sprintu*. U osvrtu na *sprint*, Scrum tim analizira izvršene aktivnosti u proteklom *sprintu*, što je u *sprintu* odrađeno dobro i što je odrađeno loše te na osnovu analize planira svoj rad u budućnosti.

Slika 5 prikazuje dijagram Scrum metodologije, Scrum timove te njihove uloge, Scrum događaje te kada se oni događaju i koliko traju, Scrum artefakte te njihovu međusobnu povezanost.

Slika 5: Scrum metodologija



Izvor: https://netmind.net/en/the-past-present-and-future-of-scrum/ (21.8.2021.)

Svako programsko rješenje razvijeno Waterfall metodologijom prolazi kroz sve faze razvoja, svaka faza razvoja započinje isključivo završetkom prethodne faze, a razvoj se ne može vratiti u prethodnu fazu. To znači da ukoliko je došlo do pogreške u nekoj od početnih faza cijelo se rješenje mora se završiti kroz sve faze te, ukoliko je potrebno ponovno razvijati.

Zahtjevi korisnika se prikupljaju i dokumentiraju na početku razvoja i koriste se za razvoj programskog rješenja. Nakon završetka početne faze, konačni korisnik nije više nikako

uključen u razvoj rješenja. Programsko rješenje se korisniku predstavlja tek u završnoj fazi razvoja, što potencijalno može dovesti do korisnikovog nezadovoljstva konačnim proizvodom.

Za razliku od Waterfall metodologije, Scrum je vrlo fleksibilan. Razvoj programskog rješenja ne prolazi kroz predefinirane faze. Programsko rješenje se najčešće konstantno iterira, nove mogućnosti se planiraju i dodaju, postojeće se ispravljaju ili nadopunjavaju. Konačni korisnik je uključen u razvoj programskog rješenja cijelo vrijeme u vidu Vlasnika proizvoda (eng. product owner) koji zajedno s ostalim članovima Scrum tima sudjeluje u izradi programskog rješenja. Zahtjevi korisnika se stalno prikupljaju i razvoj se može mijenjati u skladu s njima. S razvojem rješenja se započinje odmah, zato što nema potrebe za dugotrajnim sastancima i planiranjima koji su karakteristični za početne faze razvoja Waterfall metodologijom. Programsko rješenje ili njegovi dijelovi se isporučuju na kraju svakog sprinta sukladno zahtjevima korisnika, a korisnik može odlučiti ne koristiti programsko rješenje nastalo kao inkrement sprinta nego čekati da mu se nakon dodatnih sprintova isporuči programsko rješenje s kojim je zadovoljan. U Waterfall metodologiji uključenost krajnjeg korisnika u razvoj programskog rješenja prestaje nakon početne faze. Krajnji korisnik nema mogućnost odluke o povratku u neku od prethodnih faza životnog ciklusa razvoja programskog rješenja kako bi se nedostaci rješenja ispravili ili kako bi se dodale nedostajuće funkcionalnosti.

Prednost Scrum metodologije je stalna suradnja s krajnjim korisnikom i stalno dobivanje povratnih informacija od krajnjeg korisnika. Uloge u timu su podijeljene što omogućava timu da obavi više posla u manje vremena. Scrum timovi moraju biti iskusni i motivirani, a ukoliko nisu to može uzrokovati probijanje rokova i/ili loše programsko rješenje kao rezultat njihovog rada.

8. Usporedba DevOps i Scrum metodologije

U tradicionalnom Scrum development timu, svaki član tima obavlja aktivnosti iz svojeg područja stručnosti. DevOps inženjeri rade na aktivnostima razvoja programskog rješenja kao i na aktivnostima IT operacija. Development timovi su manji zato što poslove više programera obavlja jedna osoba.

Osim alata za pomoć pri razvoju programskog rješenja te Scrum alata, DevOps timovi koriste i različite DevOps alate i procese koji im pomažu u automatizaciji *Continuous integration* (CI) / *Continuous delivery* (CD) aktivnosti.

U Scrumu, inkrement se krajnjem korisniku isporučuje isključivo na kraju *sprinta*. Krajnji korisnik može odlučiti hoće li koristiti programsko rješenje. Za razliku od toga, u DevOpsu, programsko rješenje kontinuirano se isporučuje korisniku. DevOps timovi stavljaju prioritet na brzinu i učestalost isporuke. Programsko rješenje se testira prije završetka *sprinta* odnosno prije isporuke inkrementa. U DevOpsu programsko rješenje se testira kontinuirano, nakon svake izmjene kako se krajnjem korisniku ne bi isporučilo neispravno programsko rješenje ili programsko rješenje koje ne udovoljava njegovim zahtjevima.

Aktivnosti testiranja programskog rješenja i isporuke programskog rješenja se obavljaju ručno dok DevOps timovi automatiziraju ove aktivnosti korištenjem različitih alata.

Tablica 1 prikazuje razlike Scruma i Devopsa. Scrum se fokusira isključivo na *software development* dok se DevOps fokusira na *software development* i na *IT Operations.* U Scrumu se programsko rješenje isporučuje na kraju *Sprinta,* dok se u DevOpsu isporučuje kontinuirano. Procesi koji se odvijaju prilikom razvoja programskog rješenja u Scrumu nisu automatizirani, dok u DevOpsu jesu. Razvojni timovi koji se za razvoj programskog rješenja služe Scrum metodologijom u pravilu su veći zato što se svaki član tima specijalizira samo za jedno područje djelovanja. Svaki *developer* bavi se isključivo aktivnostima iz svog područja djelovanja, dok u DevOpsu pojedini *developer* radi na aktivnostima iz više različitih područja djelovanja. DevOps timovi u pravilu su manji, a pojedini *developeri* specijaliziraju se za više područja djelovanja.

Tablica 1: Vizualni prikaz razlika Scruma i DevOpsa

Razlike	Scrum	DevOps
Područje djelovanja	Fokus na *software development*	Fokus na *software development* i na *IT Operations*
Isporuka programskog rješenja	Programsko rješenje isporučuje se na kraju *Sprinta*	Programsko rješenje isporučuje se kontinuirano
Automatizacija	Procesi se ne automatiziraju	Procesi su automatizirani
Timovi	Veći timovi, svaki *developer* bavi se isključivo aktivnostima iz svog područja djelovanja	Manji timovi, jedan *developer* specijaliziran za više područja djelovanja

Izvor: https://www.guru99.com/agile-vs-devops.html (10.9.2021.)

9. Continuous integration (CI) / Continuous delivery (CD)

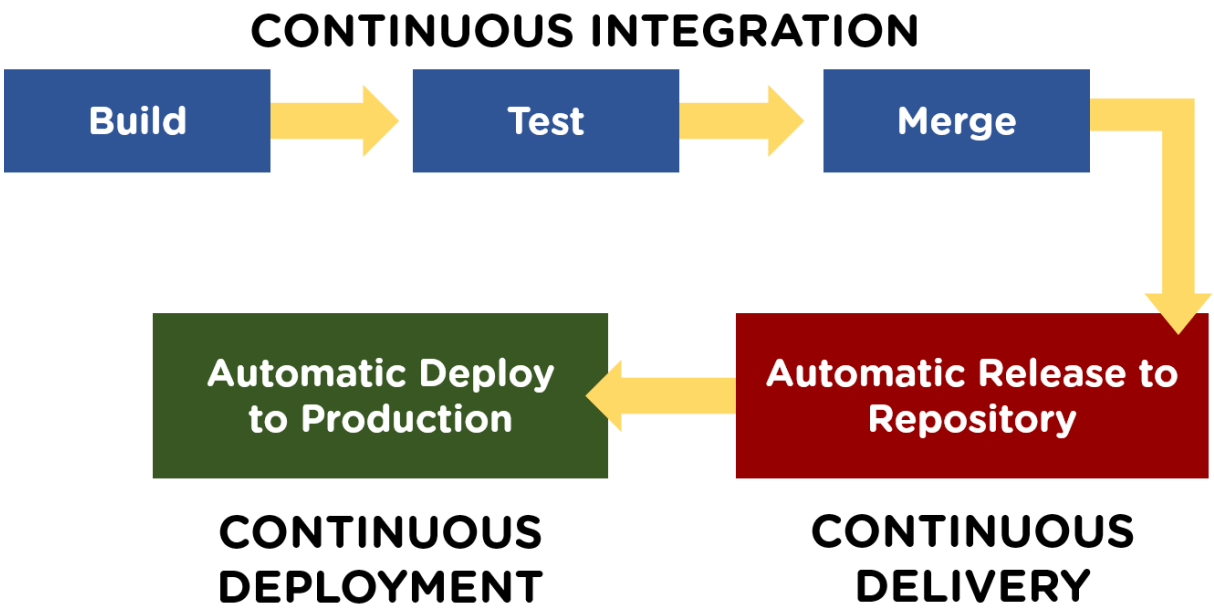
Continuous integration (CI) praksa je u razvoju programskih rješenja koja pomaže programerima u svakodnevnom radu. Tako na primjer mogu pisati dio po dio programskog rješenja ili uređivati kôd pojedine funkcionalnosti rješenja i svoje promjene postaviti na distribuirani sustav za upravljanje izvornim kôdom. Nakon svake promjene postavljene na udaljeni repozitorij na distribuiranom sustavu za upravljanje izvornim kôdom, *build management system* (npr. Jenkins ili Travis CI) kreira *build* i testira ga. *Build* sadrži programsko rješenje i sve potrebno za njegovo izvođenje i uporabu. Ukoliko rezultati testa nisu uspješni sustav će obavijestiti programera o tome, te on može napraviti potrebne izmjene kako bi njegov kôd uspješno prošao testiranje. Ovo omogućuje programerima da više vremena posvete kôdu programskog rješenja, umjesto da ga potroše na *buildanje* i testiranje. Također, ovo omogućuje dostupnost u svakom trenutku najnovije verzije programskog rješenja koja se može isporučiti korisniku u svakom trenutku.

Continuous delivery (CD) je praksa u razvoju programskih rješenja, kojom DevOps programer ili tim isporučuje krajnjem korisniku programsko rješenje na pojednostavljen način koji omogućuje veću pouzdanost i uštedu vremena. Uspješni rezultati *Continuous delivery* procesa se automatski isporučuju korisniku u predefiniranim vremenskim razdobljima, što omogućava krajnjem korisniku dostupnost uvijek najnovije verzije programskog rješenja.

Continuous integration (CI) / *Continuous delivery* (CD) *pipeline* predstavlja seriju koraka koji se moraju izvesti kako bi se programsko rješenje isporučilo krajnjem korisniku.

Slika 6 prikazuje korake procesa *Continuous integration* (CI) / *Continuous delivery* (CD). Koraci prikazani na slici su *build*, test, *merge*, isporuka aplikacije u udaljeni repozitorij te isporuka aplikacije krajnjem korisniku. U koraku *build*, programski kôd aplikacije se, ukoliko je to potrebno, prevodi (eng. compile). U slijedećem koraku, test, programsko rješenje se testira. U koraku *merge*, ukoliko je test programskog rješenja rezultirao uspjehom, promjene iz *brancha* na kojem je napravljena izmjena kôda spajaju se u glavni *branch* na udaljenom repozitoriju. U koraku isporuke aplikacije, na udaljeni repozitorij se sprema rezultat *builda*, kako bi programsko rješenje bilo spremno za isporuku krajnjem korisniku. U posljednjem koraku programsko rješenje se isporučuje krajnjem korisniku.

Slika 6: Koraci procesa *Continuous integration* (CI) / *Continuous deployment* (CD)

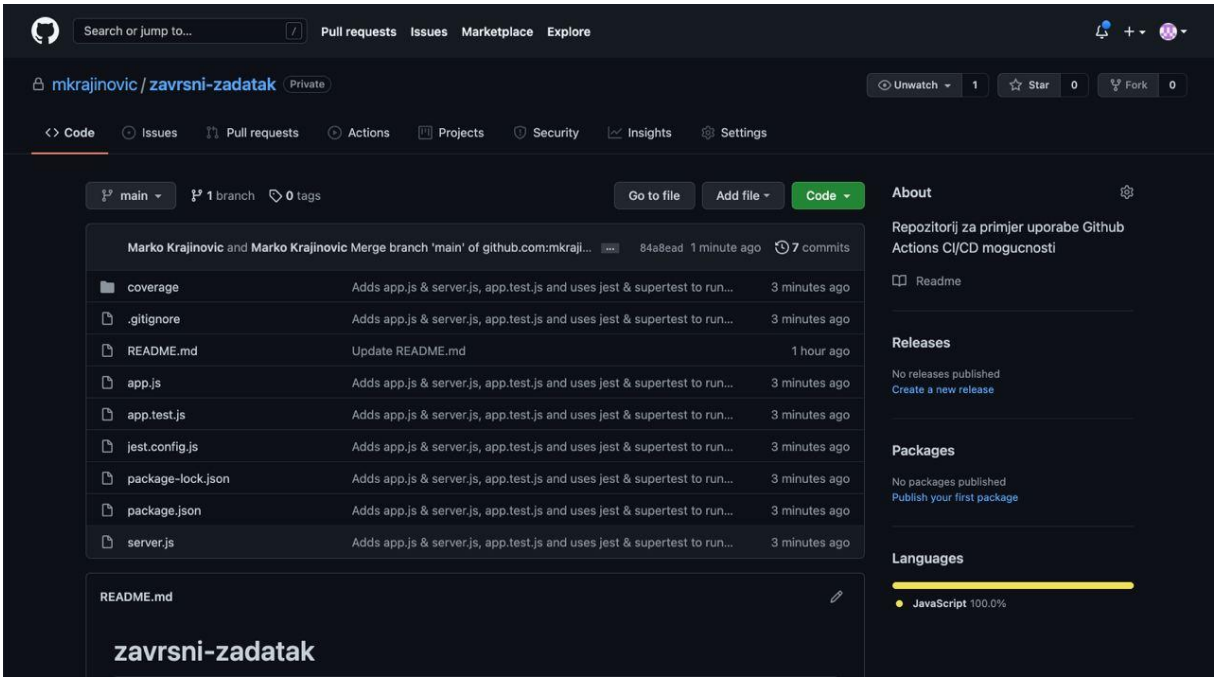


Izvor: https://www.simplilearn.com/tutorials/devops-tutorial/continuous-delivery-and-continuous-deployment *(01.09.2021.)*

Jedan od zadataka ovog rada bio je i praktično pokazati *Continuous integration* (CI) / *Continuous delivery* (CD) metode i alate. To će biti prikazano korištenjem GitHub distribuiranog sustava za upravljanje izvornim kôdom te njegove funkcije GitHub Actions. Cilj je pokazati funkcionalnosti automatskog *builda*, testiranja i isporuke jednostavnog programskog rješenja nakon svake izmjene postavljene u repozitorij programskog rješenja na distribuiranom sustavu za upravljanje izvornim kôdom GitHub.

Na slici 7 vidljivo je web sučelje distribuiranog sustava za upravljanje izvornim kôdom GitHub u kojem je vidljiv repozitorij „zavrsni-zadatak“ korisnika „mkrajinovic“.

Slika 7: Prikaz repozitorija u Web sučelju



Izvor: Autor

Ova aplikacija napisana u programskom jeziku Node.js koristi „Express“ *framework* za serviranje sadržaja u web pregledniku korisnika. Sadrži samo jednu funkcionalnost, ispis poruke u prozoru web preglednika korisnika kada korisnik pristupi putanji „/“ ove aplikacije.

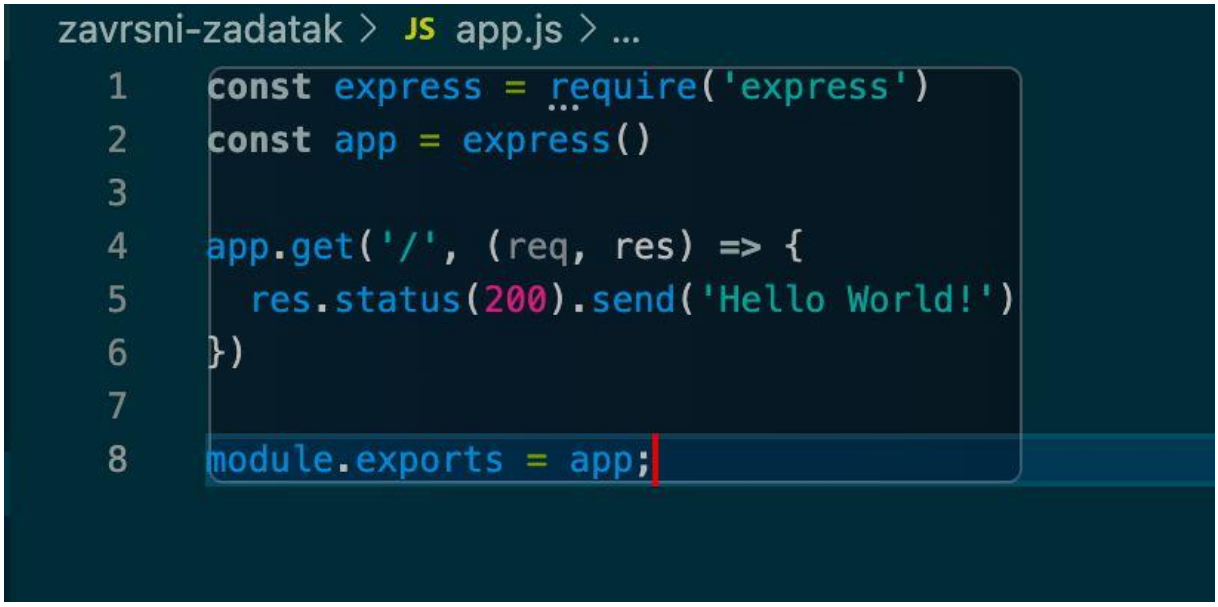
Kako bi se praktično pokazala *Continuous integration* metoda, koristi se „Jest“ *framework* i „SuperTest“ *library* za izvršavanje testova. Kako bi se pokazala *Continuous delivery* metoda, funckionalnost „Actions“ se putem SSH protokola spaja na *server* na kojem se izvršava aplikacija te pokreće skriptu deploy koja s distribuiranog sustava za upravljanje izvornim kôdom GitHub povlači najnovije promjene i ponovno pokreće aplikaciju.

Aplikacija se sastoji iz dva dijela, „app.js“, odnosno same aplikacije te „server.js“, odnosno web servera koji omogućava izvršavanje aplikacije u Web pregledniku. Programski kôd ovih dijelova nalazi se u datotekama „app.js“ i „server.js“. Strukturu aplikacije moguće je vidjeti na prethodnoj slici. Osim datoteka „app.js“ te „server.js“ i ostalih datoteka opisanih u nastavku rada, u udaljenom repozitoriju nalaze se i datoteke „.gitignore“, „README.md“, „jest.config.js“, „package-lock.json“ i „package.json“. U datoteci „.gitignore“ navedeni su direktoriji i datoteke koje se nalaze lokalno, a koje sustav Git ne smije slati na udaljeni

repozitorij. Datoteka „README.md“ sadrži opis aplikacije koji se prikazuje u Web sučelju sustava Git. Datoteka „jest.config.js“ sadrži konfiguraciju *frameworka* „Jest“. Datoteka „package-lock.json“ sadrži popis svih dodatnih programa, *librarya i frameworka* potrebnih za izvršavanje aplikacije, zajedno s njihovim verzijama. Ovo omogućava instalaciju uvijek unaprijed određene verzije dodatnih programa, *librarya i frameworka,* kako instalacija novije ili starije verzije ne bi uzrokovala poteškoće u izvršavanju ili ne mogućnost izvršavanja aplikacije. U datoteci „package.json“ navedeni su osnovni podaci o aplikaciji, njeno ime, verzija i opis, poveznica na udaljeni repozitorij aplikacije, ime autora i drugi podaci. Osim toga navedena je i glavna (eng. main) datoteka programskog kôda aplikacije. Opisane su skripte koje je moguće izvršiti, s uputama za njihovo izvršavanje. Navedene su i *dependencies* odnosno dodatni programi, *libraryi i frameworci* potrebni za izvršavanje aplikacije.

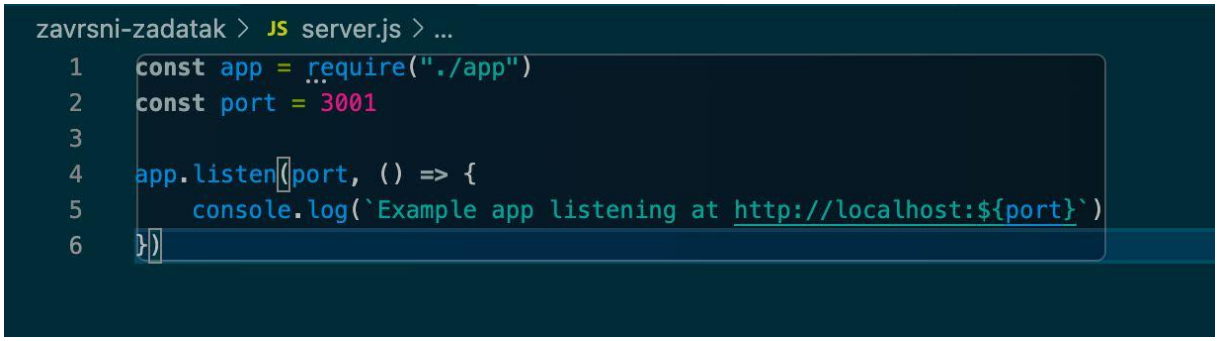
U datoteci „app.js“ uključuje se „Express“ *framework* i postavlja se varijabla „app“. „Express“ *framework* je back end *framework* za Web aplikacije pisane u programskom jeziku Node.js. Varijabla „app“ služi za poziv funkcije „express“„Express“ *frameworka.* Funkcija „app.get“ korisniku koji pristupa na „/“ putanju servera šalje HTTP status „200“ (OK), što Web pregledniku signalizira da je zahtjev uspješno izvršen te na ekranu Web preglednika ispisuje poruku „Hello World!“. U datoteci „server.js“ uključuje se datoteka „app.js“ te se postavlja konstanta „port“. Funkcija „app.listen“ pokreće Web server koji na *portu* definiranom u konstanti „port“ servira aplikaciju iz datoteke „app.js“ te u konzolu Web preglednika ispisuje poruku „Example app listening at http://localhost:port“. *Port* je dio adrese kojoj Web preglednik pristupa, a služi za preusmjeravanje Web preglednika na pojedinu aplikaciju koju Web poslužitelj poslužuje na toj adresi. Slika 8 prikazuje sadržaj datoteke „app.js“. Slika 9 prikazuje sadržaj datoteke „server.js“.

Slika 8: Prikaz sadržaja datoteke „app.js“



Izvor: Autor

Slika 9: Prikaz sadržaja datoteke „server.js“

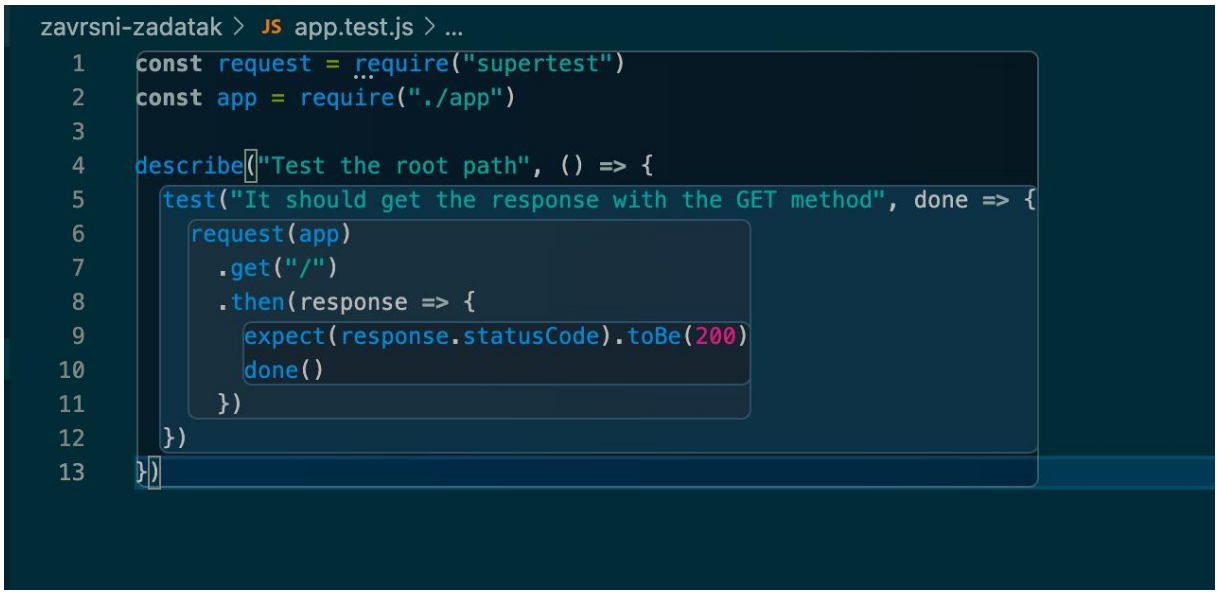


Izvor: Autor

Test koji se izvršava kao dio *Continuous integration* procesa opisan je u datoteci „app.test.js“. Na početku datoteke su definirane konstante „request“ i „app“. Konstanta „request“ poziva funkciju „supertest“ iz „SuperTest“ *librarya*. Konstanta „app“ pokreće Web aplikaciju iz datoteke „app.js“. Funkcija „describe“ izvršava test. Poziva konstantu „app“ odnosno datoteku „app.js“, koristeći GET metodu. Spaja se na „/“ putanju pokrenutog web servera te *HTTP* odgovor uspoređuje s očekivanim odgovorom (eng. response) koji je definiran kao 200 (OK). Ukoliko je provjera uspješna, test se završava te se u konzoli ispisuje poruka o

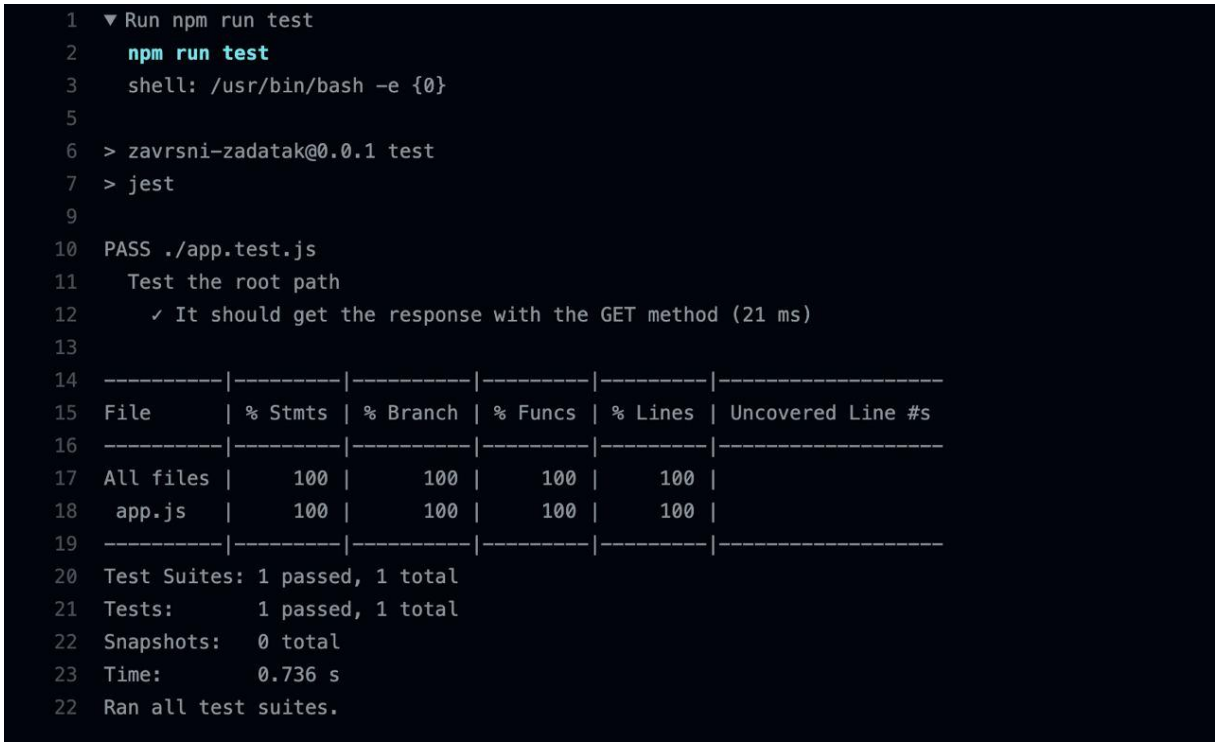
uspješno završenom testu. Ukoliko je test završio neuspješno, u konzoli se ispisuje greška zbog koje je test završio neuspješno. Slika 10 pokazuje sadržaj datoteke „app.test.js“. Slika 11 prikazuje poruku koja se ispisuje ukoliko test završi uspješno.

Slika 10: Prikaz sadržaja datoteke „app.test.js“



Izvor: Autor

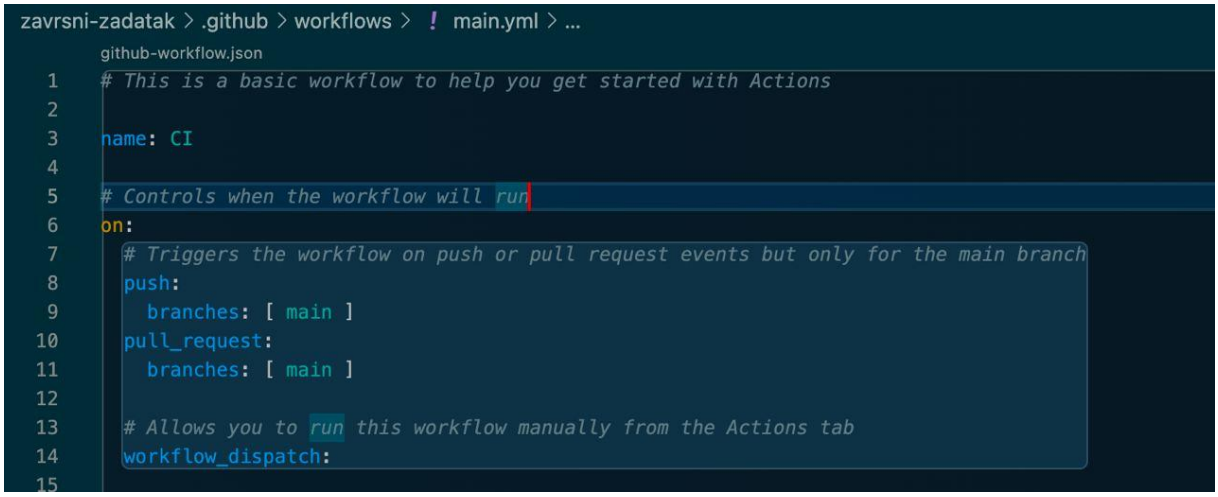
Slika 11: Poruka koja se ispisuje ukoliko test završi uspješno



Izvor: Autor

Kako bi se praktično pokazala *Continuous delivery* metoda, koristi se „Actions“ funkcionalnost GitHub distribuiranog sustava za upravljanje izvornim kôdom. Ova funkcionalnost omogućava definiranje određenih koraka koji se izvode u ovom primjeru nakon svake promjene poslane na repozitorij aplikacije na GitHub distribuiranom sustavu za upravljanje izvornim kôdom. Koraci su u ovom primjeru definirani u datoteci „main.yml“. Na početku datoteke definirano je ime akcije te kada se akcija pokreće. U ovom primjeru to se događa kada se napravi *push* odnosno kada se određena izmjena pošalje na udaljeni repozitorij aplikacije te kada se kreira *pull request* na udaljenom repozitoriju aplikacije. Slika 12 pokazuje dio datoteke „main.yml“ u kojem je definirano ime akcije te kada se ta akcija pokreće.

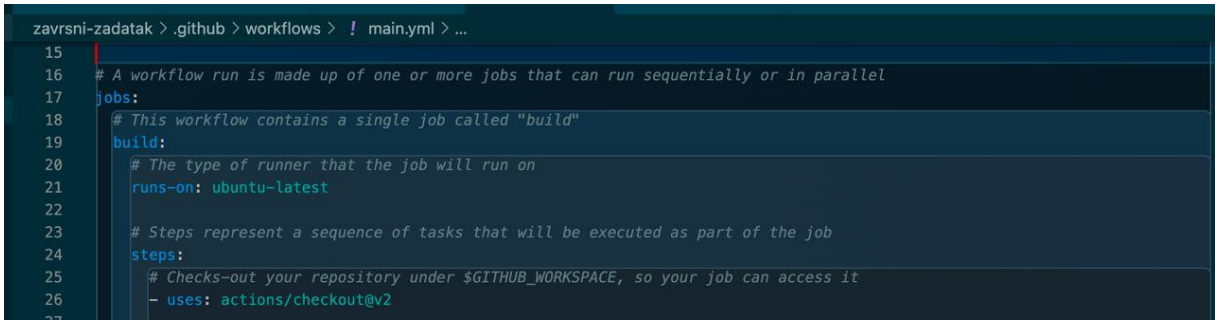
Slika 12: Dio datoteke „main.yml“



Izvor: Autor

U nastavku datoteke definirana je sekvenca zadataka (eng. Workflow) koji se izvršavaju kao dio akcije. U ovom primjeru, prvi zadatak koji se izvršava je *checkout* repozitorija aplikacije u kontejneru koji za operativni sustav koristi posljednju inačicu Ubuntu Linux operativnog sustava. Slika 13 prikazuje definiciju te sekvence zadataka.

Slika 13: Dio datoteke „main.yml“

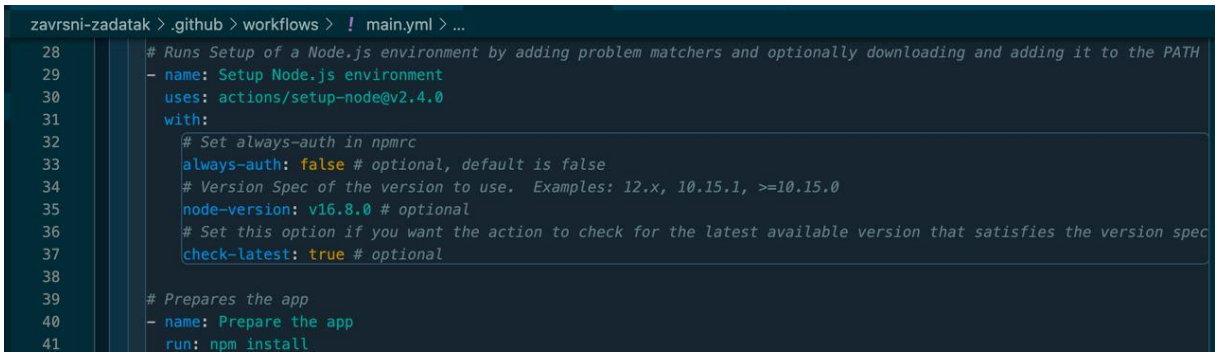


Izvor: Autor

Sljedeći definirani koraci su instalacija posljednje verzije Node *environmenta* u kojem se aplikacija izvršava te instalacija dodatnih funkcionalnosti (instalacija *frameworka* „Express“

i „Jest“ te *librarya* „SuperTest“ i ostalih pomoćnih programa) koje omogućavaju pokretanje i testiranje aplikacije. Slika 14 prikazuje definiciju ovih koraka.

Slika 14: Definicija koraka koji se trebaju izvršiti



Izvor: Autor

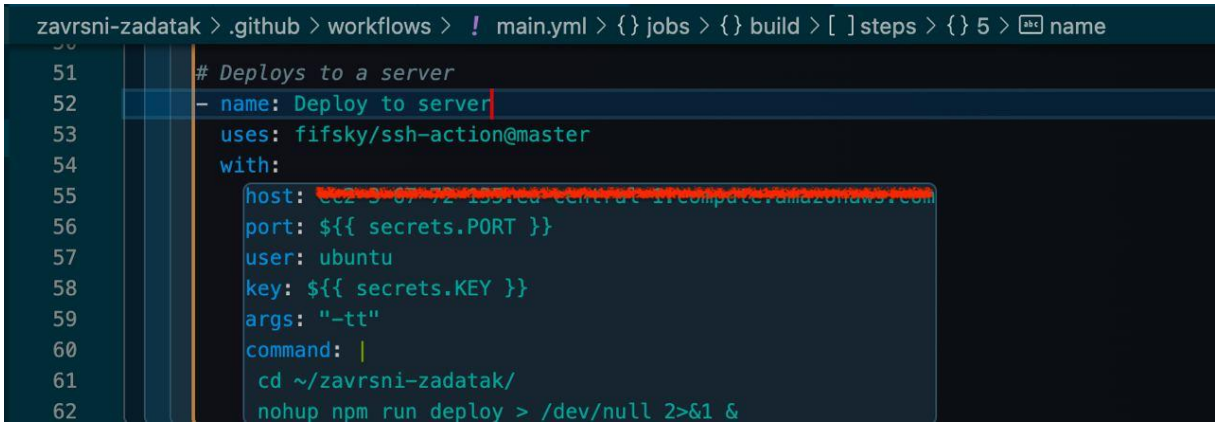
Nakon navedenog, u datoteci „main.yml“, definirani su koraci u kojima se aplikacija pokreće u pozadini te se izvršava test definiran u datoteci „app.test.js“. Slika 15 prikazuje definiciju koraka u kojima se aplikacija pokreće te se izvršava test.

Slika 15: Definicija koraka koji se trebaju izvršiti

Izvor: Autor

Posljednji definirani korak, koji se izvršava, ako su svi prethodno definirani koraci uspješno izvršeni, je isporuka aplikacije. U ovom koraku se putem protokola SSH spaja na server u oblaku, na kojem se u pozadini izvršava skripta „deploy“. Slika 16 prikazuje definiciju ovog koraka.

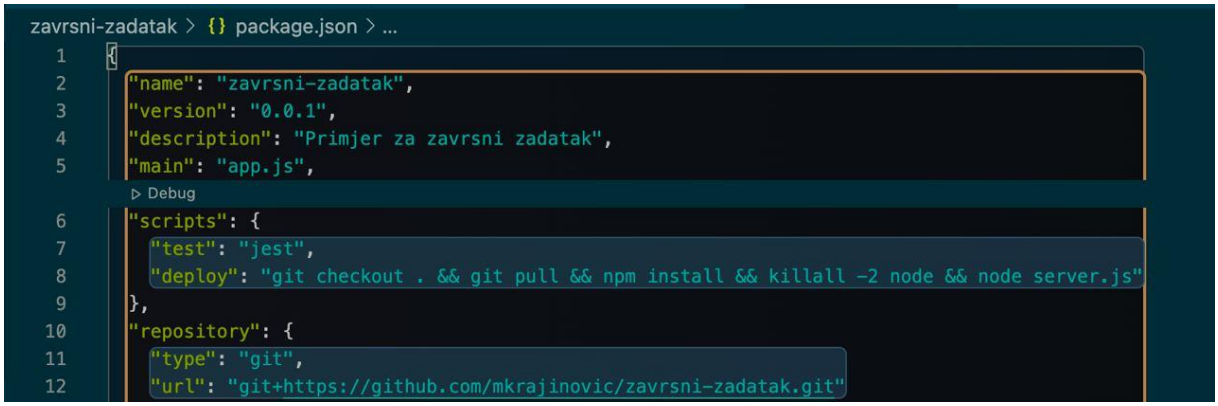
Slika 16: Definicija posljednjeg koraka



Izvor: Autor

Unutar datoteke „package.json“ definirana je skripta „deploy“. Ova skripta dohvaća najnovije promjene s udaljenog repozitorija aplikacije, instalira dodatne funkcionalnosti potrebne za izvođenje aplikacije te u pozadini ponovno pokreće aplikaciju. Slika 17 prikazuje definiciju skripte „deploy“ unutar „package.json“ datoteke.

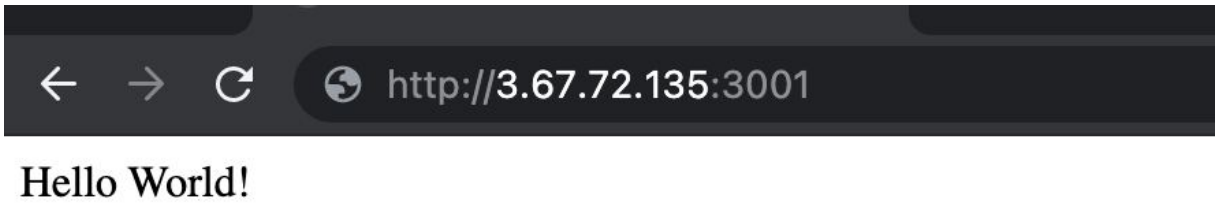
Slika 17: Definicija skripte „deploy“ unutar „package.json“ datoteke



Izvor: Autor

Kao što je navedeno ranije, aplikacija u prozoru Web preglednika korisnika ispisuje poruku „Hello World!“. Slika 18 prikazuje poruku na ekranu Web preglednika.

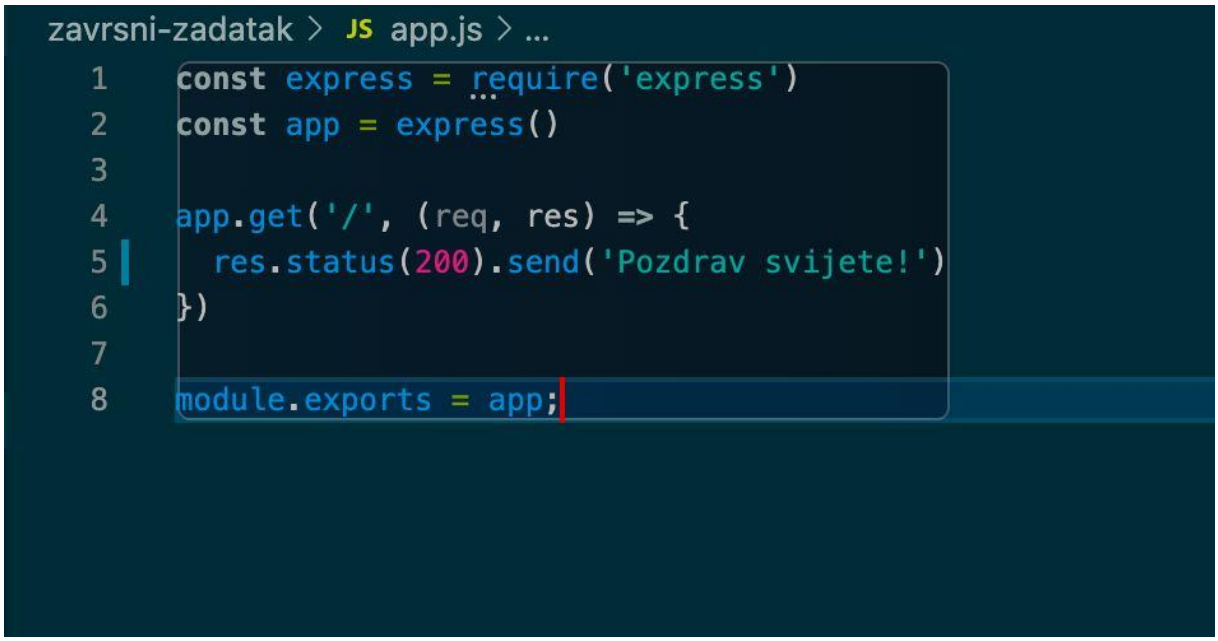
Slika 18: Prikaz poruke na ekranu Web preglednika



Izvor: Autor

Kada se u datoteci „app.js“ napravi izmjena poruke koja će se prikazivati, te se ta promjena pošalje na udaljeni repozitorij aplikacije, poziva se akcija koja je definirana u datoteci „main.yml“ koja prolazi kroz korake dohvaćanja izmjene iz udaljenog repozitorija, pripreme i pokretanja aplikacije, testiranja aplikacije te isporuke aplikacije na udaljeni server. Slika 19 prikazuje izmjenu napravljenu u datoteci „app.js“.

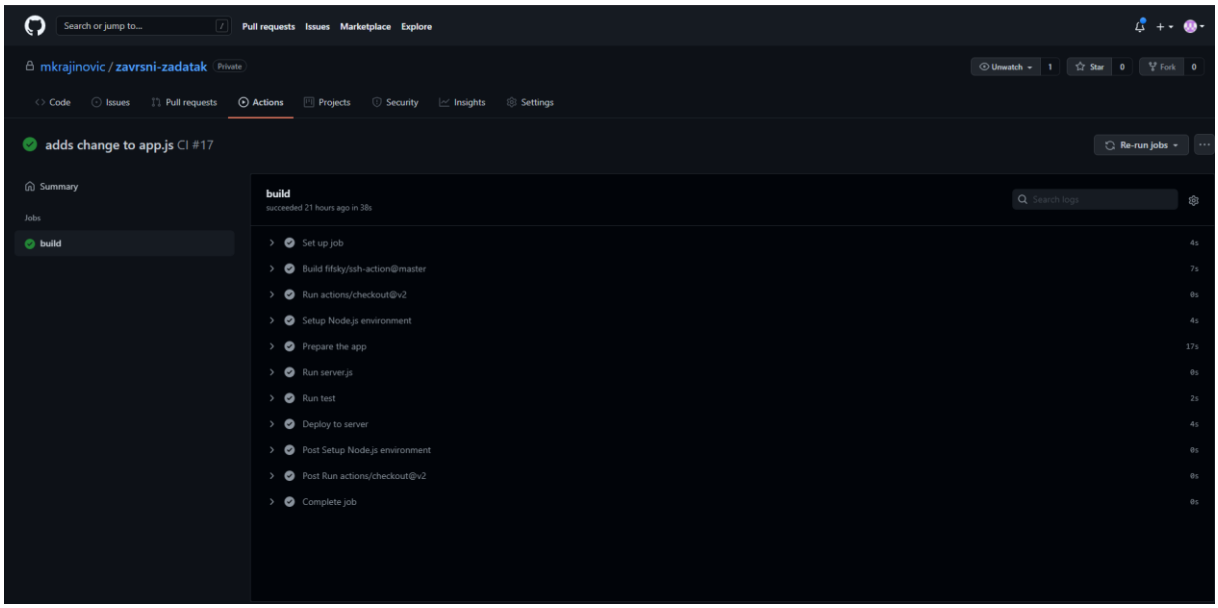
Slika 19: Izmjena u datoteci „app.js“.



Izvor: Autor

Slika 20 prikazuje izvršenu akciju u Web sučelju distribuiranog sustava za upravljanje izvornim kôdom Github.

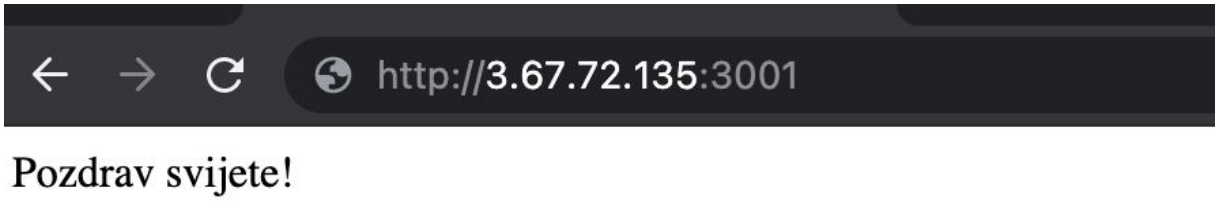
Slika 20: Izvršena akcija u Web sučelju



Izvor: Autor

Nakon izvršene akcije, ukoliko korisnik pristupi aplikaciji, u prozoru Web preglednika ispisati će se nova poruka. Slika 21 pokazuje novu poruku u prozoru Web preglednika.

Slika 21: Nova poruka u prozoru Web preglednika



Izvor: Autor

10. Zaključak

Tema ovog rada je opis DevOps metodologije razvoja programskog rješenja, s naglaskom na usporedbu s ostalim sličnim metodologijama razvoja programskih rješenja te praktični prikaz metoda stalne integracije i isporuke (eng. continuous integration / continuous delivery), što je i realizirano. U radu su objašnjene metodologije DevOps, DevSecOps, Waterfall, Agile i Scrum te je napravljena usporedba između ovih metodologija. Opisane su značajke prethodno navedenih metodologija te razlike između njih, kao i njihove prednosti i nedostatci. Također, opisane su prednosti i nedostaci korištenja svake od navedenih metoda. U drugom poglavlju opisani su DevOps alati i sustavi koji se koriste (Git, sustavi računarstva u oblaku, alati za korištenje metode kontinuirane integracije i kontinuirane isporuke, sustavi za virtualizaciju i klasterizaciju, sustavi za automatizaciju infrastrukture i alati za promatranje performansi programskih rješenja), a detaljno opisano oni najpopularniji. U posljednjem poglavlju rada praktično je prikazana metoda stalne integracije i isporuke (eng. continuous integration / continuous delivery) na primjeru jednostavne Web aplikacije. Kôd aplikacije spremljen je u udaljenom repozitoriju sustava za upravljanje izvornim kôdom GitHub. Za izvršavanje metode stalne integracije i isporuke (eng. continuous integration / continuous delivery) iskorištena je značajka „Actions“ sustava za upravljanje izvornim kôdom GitHub. Nakon izmjene u kôdu programskog rješenja, poslane na udaljeni repozitorij sustava „GitHub“, pokreće se akcija unutar sustava „Actions“, aplikacija se priprema za izvođenje, pokreće se test te ukoliko je rezultat testa uspješan, aplikacija se isporučuje na udaljeni poslužitelj unutar sustava za računarstvo u oblaku te postaje dostupna krajnjem korisniku.

Popis literature

1. https://whatis.techtarget.com/reference/The-history-of-DevOps-A-visual-timeline (07.09.2021.)
2. https://www.ibm.com/cloud/learn/devsecops (21.08.2021.)
3. https://web.archive.org/web/20200413113107/https://www.linuxjournal.com/content/git-origin-story (10.9.2021.)
4. https://cloudacademy.com/blog/disadvantages-of-cloud-computing/ (10.9.2021.)
5. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (10.9.2021.)
6. https://www.katalon.com/resources-center/blog/ci-cd-tools/ (10.9.2021.)
7. https://stackify.com/what-is-sdlc/ (17.08.2021.)
8. UNITED STATES. (1956). *Symposium on advanced programming methods for digital computers: Washington, D.C., June 28, 29, 1956*. [Washington, D.C.], Office of Naval Research, Dept. of the Navy.
9. Royce, Winston (1970), "Managing the Development of Large Software Systems"
10. https://agilemanifesto.org/iso/hr/manifesto.html (21.08.2021.)
11. https://scrumguides.org/scrum-guide.html (21.08.2021.)

Popis slika i tablica