Zbornik radova Fakulteta tehničkih nauka, Novi Sad

**UDK: 004.738.5:339**
**DOI: https://doi.org/10.24867/10BE22Ivanovic**

# AUTOMATIZACIJA TESTIRANJA SOFTVERA U OKVIRU MASTER-AGENT ARHITEKTURE

# SOFTWARE TESTING AUTOMATION WITHIN MASTER-AGENT ARCHITECTURE

Bojana Ivanović, Branko Milosavljević, *Fakultet tehničkih nauka, Novi Sad*

**Oblast – ELEKTROTEHNIKA I RAČUNARSTVO**

**Kratak sadržaj** – *U radu je opisan sistem za automatizaciju testiranja modelovan kroz master-agent arhitekturu, opisan je rad agentskog dela sistema unutar kontejnerskog okruženja. Pored implementacionih detalja istaknuti su uočeni nedostaci koji su doveli do proširenja sistema i dati su predlozi za buduće širenje istog.*

**Ključne reči:** *Automatizovano testiranje, Master-agent arhitektura, Jenkins, Allure, Docker*

**Abstract** – *Paper describes a test automation system modeled through the master-agent architecture and agent part of the system within container enviroment. In addition to implementation details, the observed shortcomings that led to the expansion of the system were highlighted with also suggestions for possible future expansion of the system.*

**Keywords:** *Test automation, Master-agent architecture, Jenkins, Allure, Docker*

## 1. UVOD

Automatizacija nam donosi velike prednosti u odnosu na manuelni rad, a to su: povećana produktivnost, efikasnija upotreba resursa, veći kvalitet proizvoda, poboljšana sigurnost, manja verovatnoća za ljudske greške, smanjenje obima posla koji obavlja čovek itd. U pogledu informacionih tehnologija, automatizacija je od ključnog značaja, pre svega zbog velikog interesa za smanjenjem verovatnoće za grešku koja dolazi sa čovekovim uključenjem u rad, ali i povećane efikasnosti i svih drugih prednosti automatizacije nad manuelnim radom.

Automatizacija je kao paradigma unela velike promene u savremeni razvoj softvera. Bilo da je taj softver deo sistema velike hidroelektrane, broda, aviona ili sistema bioskopa za rezervaciju karata, očekujemo da on radi ono što mu je i namena i to tako da radi bez grešaka. U praksi je gotovo nemoguće napisati *bug-free* softver te su prethodna očekivanja nerealna, ali je moguće napraviti dovoljno dobar softver time što će se broj problema u njemu svesti na minimum. Ovo možemo postići testiranjem softvera. Testiranje je obavezan zadatak tokom razvoja svakog softvera i onaj koji najteže pada svim programerima. Mnogi kažu da pisanje testova unosi greške u rešenje tj. softver, ali to nije ni blizu tačnog. Testiranje nam izvlači probleme na površinu, kako bismo njihovim otklanjanjem došli do „skoro savršenog“ softverskog rešenja.

---

**NAPOMENA:**
**Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.**

Izvršavanje testova napisanih za neki softver traje prilično dugo. Pokretanje stotine ili hiljade napisanih testova za neki veliki sistem, kao što je sistem hidroelektrane ili makar jednog njenog segmenta traje dugo i potrebno je da efikasnije iskoristimo to vreme. Dodatno, ukoliko je testove potrebno pokrenuti u nekom definisanom momentu, nakon spremne nove verzije softvera, na zahtev vođe tima i sl. onda taj posao pokretanja testova nije nešto što je potrebno uraditi jednom, već svaki put kada je to potrebno, a to je u praksi prilično često. Ovaj proces se može ubrzati tako što će se testiranje automatizovati. Ideja iza automatizacije testiranja bi bila da možemo pokrenuti sve testove koji su nam potrebni odjednom, bez potrebe za pojedinačnim pokretanjem svakog od testova ili grupe testova. Ono što je prednost ovakvog testiranja jeste što ne zahteva čovekovo učešće - jednom kada je pokrenuto ono se izvršava dok ne izvrši svaki od zahtevanih testova. Kako bi se izvršilo automatizovano testiranje nekog softvera, treba sagledati prvo zahteve samog softvera, preferencije tima ili kompanije i na osnovu toga odabrati najbolji alat. Sve prethodno navedeno se tiče isključenja čoveka iz nekog rada i ubrzanjem posla koji inače čovek obavlja. Jednom kada su testovi pokrenuti bitno nam je gde će se ti testovi izvršavati tj. na kom računaru i na koji način, a o tome će biti reči u nastavku ovog rada.

Ovaj rad nastao je kao rezultat rada na ideji proistekloj iz potrebe za automatizovanim testiranjem tj. kreiranjem arhitekture koja bi dovela do ubrzanja testiranja i veće produktivnosti. Takođe, ideja za kreiranjem podstaknuta je i potrebom za lakšom distribucijom rešenja klijentima koje bi za njihov konkretan softver pružilo potpuno fleksibilno testno okruženje.

## 2. TEORIJSKI KONCEPTI I TEHNOLOGIJE

### 2.1. Automatizovano testiranje

Savremeno doba je u razvoju softvera dovelo do situacija gde su timovi koji rade na razvoju softvera pod konstantnim pritiskom rokova za isporuku softvera, kao i konkretnih zahteva od strane klijenata. Ono sa čime se timovi i menadžeri susreću svakoga dana jeste da, pored poštovanja rokova, treba softver razviti uz minimalnu potrošnju resursa, pre svega novca. Kao deo činjenice da treba pružiti što više uz što manju potrošnju resursa, javlja se i to da kompanije žele da adekvatno testiraju softver koliko god je to moguće brzo, ali i temeljno. Da bi se taj cilj postigao, potrebno je okrenuti se automatizovanom načinu testiranja [1]. Može se reći da automatizovano testiranje predstavlja upravljanje i izvršavanje testnih radnji, uključujući razvoj i izvršavanje skripte za testiranje kako bi se potvrdili zahtevi za testiranjem pri čemu se koristi neki od alata za

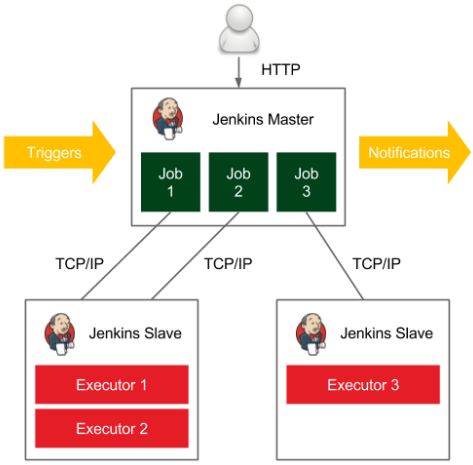
automatsko testiranje. Automatizacija testiranja svoje najveće prednosti pokazuje u situacijama gde se test skripte (ili njihovi delovi) izvršavaju često.

### 2.2. Master-agent arhitektura

Master-agent arhitektura (ranije poznata kao *master-slave*) predstavlja mehanizam koji se pre svega razvio za potrebe repliciranja baza podataka. Danas se ovaj koncept koristi u distribuiranim sistemima kao model komunikacije gde jedan uređaj ili proces (jedan od čvorova sistema) ima moć nad ostalim uređajima ili procesima. U toj komunikaciji master čvor se posmatra kao kontroler čija je jedina funkcija da vrši odlučivanje kom od potčinjenih čvorova da delegira neki posao. Posao potčinjenog čvora jeste da jednom kada dobije posao koji mu je dodeljen isti i izvrši i rezultate vrati master čvoru. Jednom kada dobije podatke, master čvor u zavisnosti od potreba servira rezultate klijentu ili ih šalje na dalju obradu ili čuvanje.

### 2.3. Jenkins

*Jenkins* je samostalni server otvorenog koda (engl. *open source*), koji se koristi za automatizaciju svih vrsta zadataka koji se odnose na izgradnju, testiranje, isporuku ili upotrebu softvera, olakšavajući kontinuiranu integraciju i isporuku softvera. Zasnovan je na serveru koji radi u servletskim kontejnerima kao što je Apache Tomcat.
Kod Jenkins-a master-agent koncept igra bitnu ulogu u delegiranju poslova. Slika 1 prikazuje delegiranje poslova od strane master čvora ka agentskim čvorovima. Kako bi se ovaj koncept implementirao, potrebno je definisati čvor koji će biti agent i kome će posao biti delegiran od strane mastera, gde agentskih čvorova može biti bezbroj. Kod agent čvora je potrebno definisati dodatne alate (engl. *plugin*) koje će agentski čvor direktno koristiti u obavljanju dodeljenog posla, ukoliko su mu oni potrebni. Pod master čvorom podrazumeva se Jenkins serverska aplikacija koja koordiniše definisanim agentskim čvorovima.




Slika 1. *Jenkins-ov master-agent koncept delegiranja poslova [3]*

Upotrebom Jenkins-a moguće je definisati različite poslove (engl. *job*) koji se mogu pokrenuti u bilo kom trenutku. *Job*-ovi su središte Jenkins *build* procesa. Na njih se može gledati kao na posebne zadatke ili korake celokupnog *build* procesa. Ovaj zadatak može da predstavlja samo kompajliranje izvornog koda ili pokretanje unit testova. S druge strane, *job* ne mora biti tako jednostavan. On može da meri pokrivenost koda testovima, primenjuje na izvornom kodu metrike kojima meri kvalitet napisanog koda, takođe može da generiše tehničku dokumentaciju ili čak izvrši *deploy* aplikacije na veb server [2].

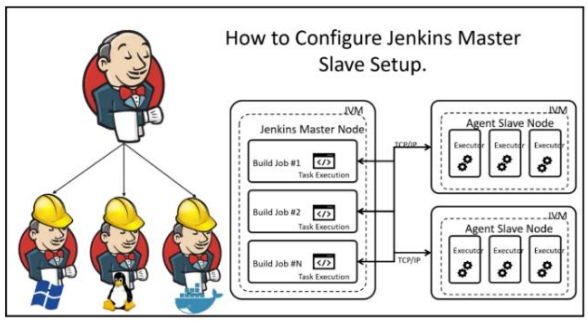
### 2.4. Allure

Allure radni okvir se razvija kao softver otvorenog koda i njegova uloga je da kreira izveštaj o testiranju koji bi bio jasan, ali sa dovoljno detalja o rezultatima izvršenih testova**Error! Reference source not found.**. Ovaj radni okvir je fleksibilan alat za generisanje izveštaja o testiranju, koji pruža koncizan izveštaj svega što je testirano i omogućuje svima koji rade na razvoju konkretnog softvera da izvuku što više korisnih informacija. Pored dosta korisnih informacija, Allure rezultate prikazuje i putem različitih grafičkih elemenata.

## 3. MODEL SISTEMA

Sistem za automatizovano testiranje baziran na master-agent arhitekturi sastoji se iz master čvora koji se nalazi unutar oblaka (engl. *cloud*), dok je agentski čvor fizička mašina. Ova arhitektura je implementirana upotrebom *Jenkins* tehnologije. Master čvor, delegira posao agentskim čvorovima, gde je za potrebe ovog rešenja bio potreban jedan agentski čvor. Sam agentski čvor izvršava dodeljeni posao i u slučaju ovog sistema taj posao izvršava se na fizičkom računaru koji komunicira sa masterom preko TCP/IP protokola. Na računaru gde će se obavljati posao koji dobije agentski čvor može biti bilo koji operativni sistem, što i prikazuje Slika 2.

U slučaju ovog konkretnog sistema za automatizaciju testiranja, na mašini koja obavlja posao nalazi se kontejnerizovana aplikacija. Kontejnerizacija je izabrana zato što je potrebno da automatizovano testiranje bude izvršeno u strogo kontrolisanom okruženju koje se može konfigurisati tako da nije potrebno vršiti nikakva dodatna podešavanja na mašini domaćinu. Ono što je velika prednost kontejnerizovane aplikacije, a što je jedan od razloga za njenu upotrebu, jeste činjenica da se takva aplikacija lakše distribuira korisnicima i u tu svrhu je upotrebljen Docker.

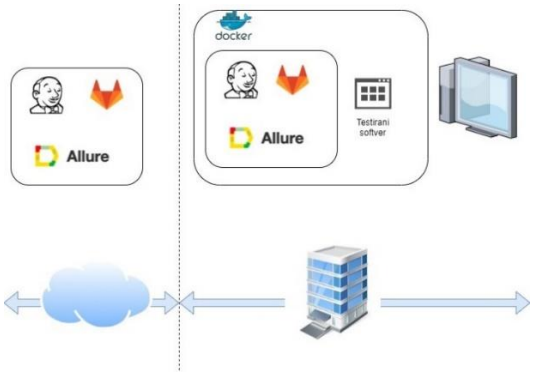



Slika 2. *Master-agent arhitektura sa prikazom različitih tipova agenata [4]*

## 4. IMPLEMENTACIJA

Sistem za automatizovano testiranje softvera u okviru master-agent arhitekture, implementiran za potrebe ovog rada, opisuje Slika 3, gde se može videti da je arhitektura takva da postoji *cloud* deo sistema, gde se nalazi Jenkins master čvor koji kontroliše obavljanje poslova koje vrši agent koji je predstavljen kao Docker kontejner. Tok testiranja je takav da master delegira posao za testiranje,

koji uključuje testove smeštene na Git repozitorijumu i ovi testovi se preuzimaju na agentsku mašinu gde se vrši testiranje. Jednom kada se testiranje završi, kao izlaz iz procesa generiše se izveštaj upotrebom Allure radnog okvira. Ovaj izveštaj se prosleđuje nazad do master čvora, gde korisnik može da ga pregleda.




Slika 3. *Arhitektura sistema za automatizovano testiranje softvera*

### 4.1. Kreiranje agentskog čvora

Kreiranje agentskog čvora moguće je izvršiti preko Jenkins aplikacije, postavljanjem osnovnih informacija o čvoru. Ono što se ističe jesu informacije potrebne za rad agenta, poput direktorijuma koji će se koristiti za rad samog Jenkins-a, zatim načina na koji će se koristi konkretan čvor, gde je u slučaju ove implementacije odabrana opcija da se on koristi maksimalno. Podešavanja agentskog čvora koja se tiču načina njegovog pokretanja zavise od načina povezivanja agentskog čvora sa masterom. U implementiranom rešenju odabran je način pokretanja agenta povezivanjem na master.
Ova opcija je zahvalnija za korišćenje u odnosu na preostale zbog manje potrebe za konfiguracijom, a pored toga je i preporučena među dostupnim opcijama za korišćenje. Pored načina pokretanja, ono što može biti važno za agentski čvor jesu alati koje on treba da koristi. Ono što je bilo potrebno za agentski čvor korišćen u ovom rešenju jeste Git, kako bi agentski čvor mogao da preuzme testove sa GitLab repozitorijuma. Da bi Git bio dostupan kao jedan od alata prilikom ove definicije, potrebno je da postoji kao jedan od *plugin*-a i da bude definisan kao globalni alat na nivou cele Jenkins aplikacije. Pored podrške za rad sa Git alatom, za potrebe ovog rešenja bitan je i rad sa Allure radnim okvirom radi generisanja finalnog izveštaja o testiranju, čiji je plugin takođe potrebno dodati na nivou cele Jenkins aplikacije.

### 4.2. Definisanje posla

Potreba za definisanjem zadatka u ovom rešenju bila je ta da se pokrene izvršavanje testova i kao izlaz tog testiranja servira Allure izveštaj o testiranju. Takođe, zadatak bi trebalo da pre pokretanja testova preuzme testove sa repozitorijuma gde se oni čuvaju. Postoji više načina za definiciju posla, a u implementiranom rešenju isprobani su *Pipeline* i *Free project.*

Prvo je isproban *Pipeline* mehanizam za definisanje poslova, ali se od istog odustalo. Ono što je loša strana ovog pristupa jeste skripta preko koje se definiše zadatak. Ona vremenom i širenjem zahteva raste i postaje nepregledna, te iako je reč o programskom kodu koji se piše unutar skripte, ipak nije jednostavno za održavanje na duže staze. Ono što se nametnulo kao dodatni zahtev koji bi dodatno automatizovao testiranje jeste da se pre pokretanja testova oni preuzmu sa repozitorijuma za čuvanje koda i smeste na neku lokaciju i nad tim preuzetim testovima pokrene izvršavanje. Ovaj zahtev nije mogao u potpunosti da se uklopi sa *Pipeline* mehanizmom, tj. Git kao alat nije bilo moguće integrisati u ovaj način rada.

Nakon isprobanog *Pipeline* mehanizma i uviđanja nedostataka za implementirano rešenje, izabran je *Freestyle project* koji predstavlja u Jenkins-u način definicije posla tj. zadatka sa mnogo širim setom mogućnosti nego što je bio slučaj kod *Pipeline*-a. Pored toga, ovaj tip zadatka tj. forma za njegovo kreiranje je jednostavnija i preglednija sa stanovišta korisničkog interfejsa, lakše je za rad i shvatanje čak i za osobe koje nisu programeri ili testeri.

Ono što je bila jedna od prepreka *Pipeline*-a jeste upotreba alata za kontrolu verziju i to je ono što *Source Code Management* sekcija rešava. Kako bi bilo moguće preuzeti kod sa repozitorijuma, potrebno je u okviru pomenute sekcije navesti putanju do repozitorijuma i izabrati način autentifikacije. Moguće je autentifikaciju vršiti koristeći GitLab kredencijale ili putem *SSH*, gde su u implementi-ranom rešenju isprobana oba ekvivalentna načina. Ostale sekcije definicije posla predstavljaju korake u kojima se definiše naredba za pokretanje testiranja i definisanje direktorijuma za smeštanje izveštaja o testiranju i one su identične u oba načina definicije posla.

### 4.3. Jenkins unutar Docker kontejnera

Kako je jedan od ključnih zahteva bio da sistem za automatizaciju testiranja radi unutar kontejnera, potrebno je omogućiti rad Jenkins-a unutar kontejnera. S jedne strane, da bi čvor imao ponašanje mastera potrebno je da sadrži celu Jenkins aplikaciju kako bi pružio sve funkcionalnosti. Ovo se može omogućiti pakovanjem Jenkins-a unutar Docker-a na dva načina: upotrebom Jenkins-a kao *base image*-a ili pakovanjem *jenkins.war*. Dok, ako je potrebno da se čvor ponaša kao agent unutar kontejnera, potrebno je obezbediti agentsko ponašanje, pakovanjem *agent.jar* unutar Docker-a.

Kada je potrebno da se čvor ponaša i kao agent i kao master tj. da bude u stanju da izvršava poslove kao agent, ali da bude master za druge agentske čvorove, potrebno je spojiti dva prethodna pri-stupa, te čvor mora posedovati i *jenkins.war* i *agent.jar*. Za potrebe implementiranog rešenja kreiran je jedan Docker *image* koji ima ponašanje i mastera i agenta, ali i jedan agentski *image* i oba Docker *image*-a su testirana tj. korišćena tokom evaluacije implementiranog rešenja.

### 4.4. Testiranje upotrebom Pytest radnog okvira

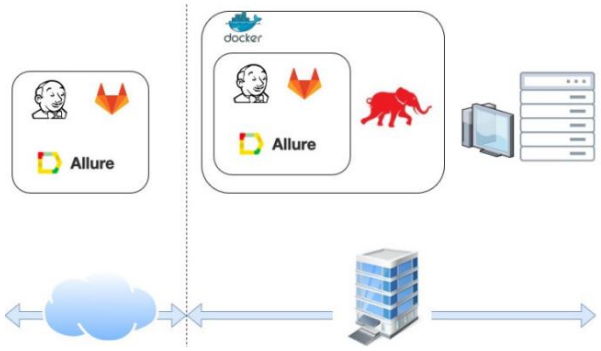
U ranoj fazi implementacije cilj je bio da u master-agent arhitekturi agentski čvor izvršava testove napisane kroz Pytest radni okvir i u početku je testirani set činilo tek nešto preko desetak testova. Svaki od tih testova bazirao se na jednostavnim tvrdnjama (asertacijama). Cilj ovih malobrojnih i jednostavnih testova bio je u tome da se

isproba ceo sistem i tok, od pokretanja posla na masteru do izvršavanja i generisanja izveštaja o istom.

### 4.5. Testiranje softvera u okruženju simulacije u realnom vremenu

Početna ideja ovog rada bila je napraviti okruženje za testiranje koje radi u master-agent režimu za konkretan softver kompanije Typhoon HIL i to unutar sistema čiji je model ranije opisan. Ideja je bila integrisati sistem za automatizovano testiranje da radi sa konkretnim softverom kompanije i bibliotekom TyphoonTest koju on koristi. Ova biblioteka je zasnovana na Pytest radnom okviru koji se koristio u početnoj fazi razvoja i predstavlja dodatak na pomenuti radni okvir. Kompanija koristi ovu biblioteku za izvršavanje testova koji su pisani za testiranje sistema u realnom vremenu. Da bi se testirali sistemi u realnom vremenu, ostvaruje se saradnja sa simulatorom gde će se praktično test izvršiti i vratiti rezultat. TyphoonTest je deo TyphoonTest IDE, a koji je deo THCC (*Typhoon HIL Control Center*) softvera. Takođe, kako je na kraju potrebno prikazati izveštaj o testiranju u vidu Allure izveštaja korišćen je i typhoon-allure koji je deo THCC-a i predstavlja *wrapper* oko samog Allure radnog okvira.

Implementirano rešenje radi u master-agent arhitekturi, gde se master čvor nalazi unutar oblaka. Kako je ovaj sistem kreiran za domenski softver kompanije Typhoon HIL, njihov kontrolni centar je spakovan unutar kontejnera koji predstavlja agentski čvor. Slika 4 prikazuje implementirani sistem, gde je kontejnerizovan THCC softver.




Slika 4. *Arhitektura sistema za automatizovano testiranje Typhoon HIL softvera*

### 4.6. Proširenje sistema

Tokom rada na implementaciji projektnog rešenja, došlo se do zaključka da je činjenica o tome da korisnik ne vidi nikakve rezultate sve dok se testiranje ne završi, dosta problematična. Samo testiranje može da traje nekoliko sekundi, ali i do nekoliko sati. Ono što je mana ovog sistema, kada je reč o prikazu rezultata, jeste to što su rezultati vidljivi samo na masteru. Master čvor tj. Jenkins aplikacija u okviru koje će biti serviran Allure izveštaj o testiranju uglavnom nije dostupna korisniku ili možda ne svim korisnicima, što znači da ukoliko korisnik nema pristup masteru ne može da vidi rezultate. Pored vizuelizacije rezultata, javlja se i problem čuvanja Allure izveštaja, koji zapremaju dosta memorijskog prostora, jer su sačinjeni od mnogobrojnih *.json* datoteka. Njihovo čuvanje na duže staze predstavlja ozbiljan problem.

Zbog pomenutih razloga celokupan sistem je proširen tako što je kreiran još jedan Docker kontejner u koji su smeštene dve aplikacije: Django aplikacija koja nadgleda testiranje i Angular aplikacija koja vrši prikaz pređašnjih rezultata testiranja i pruža korisniku bolje iskustvo, ali korisnik i dalje ne vidi u realnom vremenu rezultate što je mana. Takođe, kao serverska aplikacija koja radi sa podacima, podignuta je Spring boot aplikacija koja radi sa MySQL bazom podataka u kojoj se pamte informacije o testiranju i ova aplikacija se nalazi unutar oblaka.

## 5. ZAKLJUČAK

U radu je opisan sistem za automatizaciju testiranja modelovan kroz master-agent arhitekturu, gde je opisan rad agentskog dela sistema unutar kontejnerskog okruženja, kao i primena ovog sistema za automatizaciju testiranja sistema u realnom vremenu. U glavnom poglavlju o implementaciji je prikazan celokupan implementirani sistem.

Kao neki predlog budućeg razvitka ovog rada jeste omogućiti korisniku uvid u rad sistema u realnom vremenu. Takođe, ovo rešenje je isprobano, tako što su pokretani skupovi testova reda veličine do par stotina testova. Bilo bi interesantno ovo rešenje primeniti za testiranje nekog velikog sistema poput sistema automobila, aviona ili hidroelektrane sa količinom testova gde red veličine skupa testova ide do nekoliko hiljada testova i posmatrati ponašanje sistema.

## 6. LITERATURA

[1] Dustin, Elfriede, Jeff Rashka, and John Paul. *Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance*. Addison-Wesley Professional, 1999.

[2] Smart, John Ferguson. *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. " O'Reilly Media, Inc.", 2011.

[3] https://www.oreilly.com/library/view/continuous-delivery-with/9781787125230/assets/162a6bf6-9b46-4604-93ec-d818fbfde1f9.png (pristupljeno u septembru 2020.)

[4] https://digitalvarys.com/how-to-configure-jenkins-master-slave-setup/ (pristupljeno u septembru 2020.)

### Kratka biografija:

**Bojana Ivanović** rođena je u Novom Sadu 1996. god. Upisala je osnovne akademske studije na Fakultetu tehničkih nauka 2015. god. na smeru Softversko inženjerstvo i informacione tehnologije. Diplomirala je 2019. god. i iste godine je upisala master akademske studije na Fakultetu tehničkih nauka, smer Softversko inženjerstvo i informacione tehnologije, na usmerenju Elektronsko poslovanje.
kontakt: bojana.ivanovic@uns.ac.rs