Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/06BE25Pocuca

PREDLOG ARHITEKTURE CI/CD PIPELINE-A KORIŠĆENJEM JENKINS I DOCKER ALATA

THE PROPOSAL OF CI/CD PIPELINE USING JENKINS AND DOCKER

Milena Počuča, Srđan Popov, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su detaljno analizirana dva najpopularnija alata za pravljenje CI/CD pipeline-a. Infrastruktura je postavljena uz akcenat na Docker na kome su pokrenuti svi servisi potrebni za CI/CD. Za primer aplikacije na kojoj je pipeline implementiran uzeta je SpringBoot i AngularJS aplikacija podsistema za fakturisanje.*

Ključne reči: *CI/CD, Docker, Jenkins, SpringBoot, AngularJS, pipeline, DevOps*

Abstract – *The paper analyzes the two most popular tools for implementing CI/CD pipeline. The infrastructure is set up with the accent on Docker on which all the services neeeded for this pipeline are configured and run. The application on which the pipeline was implemented is a SpringBoot and AngularJS application subsystem for invoices.*

Keywords: *CI/CD, Docker, Jenkins, SpringBoot, AngularJS, pipeline, DevOps*

1. UVOD

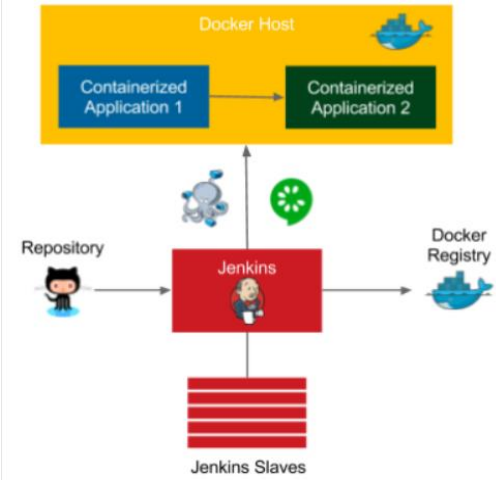
CI/CD predstavlja skup CI (*continuous integration*) i CD (*continuous delivery*) paradigmi koji je našao široku primenu u industriji. CI predstavlja pristup u kome se programeri podstiču da implementiraju male izmene na kodu što je frekventnije moguće. CD se nadovezuje na CI i automatizuje proces isporuke aplikacija na izabrana okruženja. CI/CD *pipeline* u radu je implementiran korišćenjem *Jenkins* i *Docker* alata. CI/CD *pipeline*-om bavi se *DevOps*.
Cilj rada je detaljno upoznavanje sa *Jenkins* i *Docker* alatima uz demonstraciju načina implementacije infrastrukture za CI/CD, kao i uz osvrt na već postojeće implementacije *pipeline*-a.

2. INFRASTRUKTURA SISTEMA

Jezgro sistema predstavlja *Docker* mašina koja u sebi ima pokrenute kontejnere svih alata koji su potrebni za funkcionisanje aplikacije. Na *Docker*-u su startovani na lokalnom računaru *Jenkins* kao CI server, *GitLab* kao sistem za upravljanje git repozitorijumom, *Minio* namenjen čuvanju biblioteka kreiranih od strane *Jenkins*-a, *MySQL Server* za upravljanje bazom podataka, *SonarQube* za proveru kvaliteta koda, *Redis* za konfiguraciju *Docker* registra.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Srđan Popov, vanr. prof.

Na slici 1 prikazan je primer arhitekture sistema koji se bazira na *Docker*-u.



Slika 1. *Prikaz arhitekture bazirane na* Docker-u

3. DEVOPS

DevOps predstavlja spoj tehnologija koje objedinjuju razvoj softvera (*Dev*) i IT operacija (*Ops*) kako bi skratili životni ciklus razvoja softvera**.**
Ciljevi *DevOps* paradigme su učestalije isporučivanje koda, skraćivanje vremena potrebnog da se stigne do produkcionog rešenja kao i vremena koje protekne od kreiranja zahteva do njihovog ispunjenja, snižavanje stope otkaza novijih verzija proizvoda, brže srednje vreme oporavka sistema.

4. JENKINS

Jenkins je samostalan, javni automatizacioni server koji se može koristi da automatizuje razne vrste zadataka povezane sa izgradnjom, testiranjem, isporukom ili *deployment*-om softvera. Jedna od najvećih prednosti ovog servera je velika proširivost koja omogućava da se njegove funkcionalnosti prošire instalacijom *plugin*-ova.

4.1. Pipeline

Jenkins pipeline je skup *plugin*-ova koji omogućava implementaciju i integraciju *continuous delivery pipeline*-ova u *Jenkins*. *Jenkinsfile* komitovan na repozitorijum omogućava veliki broj prednosti jer je verzionisan i u svakom trenutku su sve izmene na fajlu vidljive.

4.2. Zašto pipeline

Podržava širok skup slučajeva od jednostavih *continuous integration* do obimnih CD *pipeline*-ova. Sadrži brojne funkcionalnosti, kao što su implementacija *pipeline*-a u kodu, pauziranje, svestranost, proširivost.

4.3. Rečnik pojmova

Osnovni pojmovi u *Jenkins*-u su: *agent, artifact, build, core, downstram, executor, fingerprint, folder, item, job, label, master, node, project, pipeline, plugin, stage, step, trigger* [4].

5. SINTAKSA PIPELINE-A

Od verzije 2.5 plugin-a za *pipeline*, postoje dve sintakse za njen opis koje će u nastavku biti objašnjene.

5.1. Deklarativni pipeline

Ova sintaksa je prilično nov dodatak *Jenkins pipeline*-a koji predstavlja dosta jednostavniji način definisanja**.** Svaki pipeline ovog tipa mora biti obuhvaćen u okviru *pipeline* bloka.

5.1.1. Sekcije

Obično su sastavljene iz jedne ili više direktiva.

5.1.1.1 Agent

Sekcija agent određuje gde će se čitav *pipeline*, ili određena faza, izvršavati na *Jenkins* okruženju, u zavisnosti od toga gde je sekcija agent smeštena. Mora se definisati na najvišem nivou unutar *pipeline* bloka, ali se može definisati i unutar *stage* bloka. Sadrži parametre koji se mogu primeniti na najviši nivo *pipeline* bloka, ili unutar svake *stage* direktive. Mogući parametri su *any, none, label, node, docker, dockerfile, kubernetes* [4].

5.1.1.2. Post

Ova sekcija sadrži jedan ili više dodatnih koraka koji se pokreću nakon završetka *run* dela *pipeline*-a ili *stage*-a. Podržava brojne *post-condition* blokove koji dozvoljavaju izvršavanje koraka unutar svakog uslova. Mogući uslovi su: *always, changed, fixed, regression, aborted, failure, success, unstable, unsuccessful, cleanup* [4].

5.1.1.3. Stages

Ova sekcija sadrži sekvencu jedne ili više *stages* direktiva i označava sekciju gde je najveći deo posla u *pipeline*-u smešten. Preporučuje se da ova sekcija sadrži bar jednu *stage* direktivu za svaki deo CD procesa – *Build, Test, Deploy* [4].

5.1.1.4. Steps

Sekcija *steps* definiše skup jednog ili više koraka koji se mogu izvršavati u datoj *stage* direktivi. Sekcija je obavezna, ne zahteva parametre i dozvoljena je unutar svakog *stage* bloka.

5.1.2. Direktive

5.1.2.1. environment

Ova direktiva predstavlja sekvencu ključ-vrednost parova koji će biti definisani kao varijable okruženja za sve korake, ili za korake specifične za neki *stage*, u zavisnosti od toga gde se ova direktiva nalazi u *pipeline*-u. Podržani tipovi kredencijala: *Secret Text, Secret File, Username and Password, SSH with Private key* [4].

5.1.2.2. options

Direktiva *options* omogućuje konfiguraciju opcija karakterističnih za *pipeline* unutar samog *pipeline*-a. Dostupne opcije za pipeline: *buildDiscarder, checkoutTo-Subdirectory, disableConcurrentBuilds, disableResume, newContainerPerStage. overrideIndexTriggers, quiet-Period, retry, skipDefaultCheckout, skipStagesAfter-Unstable, timeout, timestamps, parallelsAlwaysFailFast,* Opcije koje se odnose na *stage* predstavljaju deo opcija za *pipeline:- skipDefaultCheckout, timeout, retry, timestamps* [4].

5.1.2.3. parameters

Direktiva *parameters* obezbeđuje listu parametara koji bi korisnik trebalo da navede kada pokreće izvršavanje *pipeline*-a. Tipovi parametara su: *string, text, booleanParam, choice, password.*

5.1.2.4. triggers

Direktiva *triggers* definiše automatski način na koji bi *pipeline* trebalo da se ponovo trigeruje. Vrste trigera su: *cron, pollSCM, upstream* [4].

5.1.2.5. Jenkins cron sintaksa

Jenkins cron sintaksa prati sintaksu *cron utility*-ja sa sitnim razlikama. Svaka linija se sastoji iz 5 delova koji su odvojeni TAB-om ili razmakom. U sintaksi se specificiraju 'M H DOM MONTH DOW', gde M predstavlja minut u satu, H sat u danu, DOM dan u mesecu, MONTH mesec u godini, a DOW dan u nedelji.

5.2.1.5. stage

Direktiva *stage* se nalazi u okviru *stages* sekcije. Sadrži *steps* sekciju, opcionu agent sekciju ili druge direktive specifične za *stage*. U praksi, najveći deo realnog posla *pipeline*-a se odvija unutar jedne ili više *stage* direktiva. Potrebno je definisati bar jednu *stage* direktivu koja će imati jedan obavezni parameter, a to je ime *stage*-a.

5.2.1.6. tools

Ova sekcija definiše alate koji se automatski instaliraju i dodaju na PATH. Ignoriše se ukoliko se specificira *agent none*. Dozvoljena je unutar *pipeline* ili *stage* bloka. Podržani alati su *maven, jdk* i *gradle*.

5.2.1.7 when

Direkvita *when* dozvoljava pipeline-u da odluči da li *stage* treba da se izvrši u zavisnosti od datog uslova. Unapred dostupni uslovi su: *branch, buidlingTag, changelog, changeset, changeRequest. environment, equals, expression, tag, not, allOf, anyOf, triggeredBy* [4].

5.1.5. Steps

Deklarativni *pipeline* koriste sve korake koji su dokumentovani u Pipeline Steps reference dokumentaciji i sadrže obilnu listu koraka.

5.2. Skriptovani pipeline

Za razliku od deklarativnog, skriptovani pipeline je DSL opšte namene izgrađen pomoću *Groovy*-ja. Većinu fukncionalnosti koje omoguće *Groovy* jezik moguće je koristiti u skriptovanom *pipeline*-u. Najvažniji aspekt ovog *pipeline*-a je kontrola toka.

5.3. Deklarativni naspram skriptovanog pipeline-a

Oba *pipeline*-a su fundamentalno ista u *pipeline* podsistemu. Razlikuju se ipak, u sintaksi i fleksibilnosti. Deklarativni *pipeline* ograničava šta je dostupno korisniku sa striktnijom i predefinisanom strukturom, što ga čini idealnim izborom za jednostavniji CD *pipeline*. Skriptovani *pipeline* obezbeđuje mali broj ograničenja, može se reći da su to ograničenja definisana u samom *Groovy* jeziku, ne u *pipeline*-u, što ga čini idealnim izborom za iskusnije korsnike i korisnike koji imaju kompleksne zahteve. Deklarativni *pipeline* promoviše deklarativni model programiranja, dok skriptovani *pipeline* promoviše imperativni [4].

6. DOCKER

Docker je alat koji je dizajniran da olakša kreiranje, *deploy*-ovanje i pokretanje aplikacija korišćenjem kontejnera. *Docker, Inc.* je osnovan 2010. godine, a javnosti je predstavljen 2013.

6.1. Kontejneri i virtuelne mašine

Docker kontejneri se mogu porediti sa virtuelnim mašinama. I jedni i drugi imaju sličan cilj, a to je izolacija aplikacije i njenih zavisnosti u samoodrživu jedinicu koja se može svuda pokrenuti. Virtuelna mašina ima svoj virtuelni operativni sistem u kome hipervizor igra esencijalnu ulogu tako što joj obezbeđuje platformu za upravljanje i izvršavanje *guest* operativnog sistema.
Za razliku od virtuelne mašine, kontejner obezbeđuje virtuelizaciju na nivou operativnog sistema. Kontejnerizacija je takođe vrsta virtualizacije s prednošću da je efikasnija jer nema *guest* operativnog sistema, već koristi *host*-ov operativni sistem.

6.2. Docker Engine

Docker engine predstavlja srž *Dockera*. To je aplikacija koja se instalira na *host* mašinu. Funkcioniše po principu klijent-server arhitekture. Glavne komponente ove arhitekture su: server koji predstavlja program koji se dugo izvršava zvani *daemon* proces, CLI koji prestavlja klijenta, REST API preko koga CLI i *daemon* komuniciraju [5].

6.3. Arhitektura Docker-a

Docker koristi klijent-server arhitekturu u kojoj *Docker* klijent komunicira sa Docker *daemon*-om koji ustvari radi najveći posao - izgradnju, pokretanje i distribuciju *Docker* kontejnera.

6.4. Slojevi Docker-a

Docker image se kreira iz serije slojeva (eng. *layers*) gde svaki sloj predstavlja instrukciju *image*-a u *Dockerfile*-u. Svi slojevi sem poslednjeg sloja imaju samo pristup čitanja, dok se na poslednjem sloju može i pisati.

6.5. Dockerfile

Docker automatski pravi *image* čitanjem instrukcija iz *Dockerfile*-a. Ovaj fajl je tekstualni dokument koje korisnik može da koristi da bi napravio *image*. Komanda *docker build* pravi *image*.
Dostupne instrukcije koje se zadaju su:
FROM (inicijalizuje *build* i postavlja bazni *image* za ostale instrukcije),
RUN (izvršava komande u novom sloju koji je iznad *image*-a),
EXPOSE (obaveštava *Docker*-a da kontejner sluša na određenom portu),
ENV (postavlja varijablu na zadatu vrednost),
ADD (kopira nove fajlove iz zadatog izvora i dodaje iz na fajl sistem *image*-a),
COPY (kopira nove fajlove iz izvora i dodaje ih na fajl sistem kontejnera),
VOLUME (kreira pristupnu tačku sa zadatim imenom i pamti je kao eksterni volumen *host*-a ili drugih kontejnera),
USER (postavlja korisniko ime i grupu korisnika koji će se koristiti za pokretanje *image*-a) [5].

6.6. Docker compose

Compose je alat za definisanje i pokretanje *Docker* aplikacija sa više kontejnera. Koristi *yaml* fajl za konfigurisanje servisa aplikacije. To je proces koji se sastoji iz tri koraka: definisanje okruženja aplikacije korišćenjem *Dockerfile*-a, definisanje servisa koji prave aplikaciju u *docker-compose.yml* fajlu i pokretanje *docker-compose up* komande kako bi se sve to pokrenulo u izolovanom okruženju [5].

7. ALATI ZA IZGRADNJU CI/CD PIPELINE-A

CI/CD *pipeline* se može postaviti korišćenjem većeg broja alata. Ovo predstavlja trenutno najpopularniji trend u razvoju softvera.

7.1. Jenkins

Ovo je najpopularnjie rešenje CI/CD *pipeline*-a koje je besplatno.

7.2. Microsoft VSTS

VSTS nije samo servis za CI/CD, već i repozirotijum za kod i planiranje projekata. VSTS je pravljen za *Visual Studio*, ali se može integrisati i sa drugim alatima. Besplatan je za timove do 5 članova.

7.3. Bamboo (Atlassian Dev Tool)

Bamboo je CI server koji je *Atlassian*-ov proizvod, podržava *Docker*, ima probni period od 30 dana, nakon čega se naplaćuje.

7.4. GitLab

GitLab predstavlja skup alata za upravljanje skoro svim aspektima životnog ciklusa razvoja softvera. Dostupan je kao besplatan *Community* i *Enterprise* edicija koja može biti *host*-ovana ili se nalaziti na mašini.

7.5. Codeship

U odnosu na druge alate koji pružaju UI za naprednu konfiguraciju, *Codeship* je uglavnom baziran na skriptama. Poseduje *Basic* i *Pro* verzije koje se naplaćuju.

7.6. Codefresh

Codefresh svaki korak u *pipeline*-u izvršava na posebnom kontejneru. Besplatna verzija omogućuje 3 korisnika mesečno, *Pro* verzija 10, dok *Enterprise* omogućuje neograničen broj.

7.7 TeamCity

TeamCity je CI server koji ima ugrađenu integraciju sa *Docker*-om. Dostupan je besplatno sa ograničenom licencom, a postoji i verzija koja se plaća.

7.8 Travis CI

Travis CI je *host*-ovani servis za *GitHub* projekte. Besplatan je za javne projekte i omogućuje pretplatu za privatne projekte.

7.9. GoCD

GoCD je besplatan softver koji se instalira lokalno. Ima i plugin za docker, besplatan je uz opciju plaćanja podrške.

7.10. CircleCI

CircleCI je drugačiji od ostalih alata jer podržava *build*-ovanje, testiranje i *deploy*-ovanje iOS i macOS projekata korišćenjem macOS virtuelne mašine, mada se može *build*-ovati i korišćenjem *Linux* virtuelne mašine. Cena *CicrleCI*-a zavisi od kontejnera i platforme. Linux kontejner je besplatan, ali macOS kontejneri i lokalna instalacija se plaćaju.

8. OSVRT NA POSTOJEĆE IMPLEMENTACIJE CI/CD PIPELINE-A

8.1. Automatizacija CI/CD pipeline-a za projekat baziran na agilnom pristupu

Ciljevi ovog rada su razmevanje CI/CD *pipeline*-a analizom postojećih metoda, pronalaženje odgovarajuće metode za merenje performansi postojećeg sistema, predlog mehanizma za izvršavanje testova opterećenja korišćenjem saobraćaja u produkciji sa minimalnim posledicama na sistem. U radu su od alata korišćeni *Git* kao sistem za kontrolu verzija, *Nexus* kao repozitorijum izvršivih fajlova i *Jenkins* kao CI server. Zaključeno je da je CI/CD *pipeline* poboljšao agilan proces isporučivanja i poboljšao je produktivnost sistema [1].

8.2. CI/CD za HPC korišćenjem Jenkins-a i Singularity-ja

U radu je opisano kako je *Research Computing* ogranak Bolder (eng. *Boudler*) Univerziteta u Koloradu primenio CI/CD paradigmu na RMACC *Summit supercomputer* kako bi obezbedio sistem inženjerima i istraživačima da iskoriste prednosti CI/CD paradigme.

Kao CI server, korišćen je *Jenkins* koji je povezan sa *Singularity*-jem koji predstavlja rešenje za kontejnerizaciju. Uključivanjem CI/CD paradigme u proces rada HPC-a je povećalo potencijal za isporuku visoko kvalitetnih i pouzdanih softvera [2].

8.3. Implementacija CI principa u maloj kompaniji: studija slučaja

Kao motivacioni faktori za razvoj ove paradigme u kompaniji, identifikovani su poboljšanje kvaliteta sotfvera, veća pouzdanost *release*-a, kraći *time-to-market* i poboljšanje produktivnosti programera. Centralizovano rešenje, kao što je *GitLab* je primećeno kao najprimerenije rešenje jer on spaja sve potrebne servise. Zaključeno je da je CI/CD paradigma donela poboljšanje rada kompanije, ali da je potrebno nastaviti sa radom na istoj, tako što će preostale repozitorijume migrirati na novi sistem i početi što pre sa testiranjem [3].

9. IMPLEMENTACIJA REŠENJA

Čitav sistem počiva na *Docker*-u. U njemu su pokrenuti *GitLab*, *Jenkins*, *Redis*, *SonarQube*, *Minio* i *MySQL*. Sistem radi tako što kada se kod postavi na repozitorijum, *Jenkins pipeline* registruje da je došlo do izmene repozitorijuma, povlači novi kod i pokreće *build* aplikacije. Nakon što se *build* završi, startuju se testovi i ukoliko se svi uspešno izvrše, .jar koji je napravljen se postavlja na *Minio* i šalje se u *Docker* koji kreira novi kontejner za aplikaciju. *SonarQube* vrši analizu novog koda uz proveru pokrivenosti koda testovima. Kako *Docker* ponovo pokreće kontejner aplikacije sa novonastalim izmenama, njih je moguće odmah testirati.

10. ZAKLJUČAK

U radu je prikazano kako se konfiguriše CI/CD *pipeline* za potrebe manjeg projekta uz detaljna objašnjena dva glavna korišćena alata – *Docker* i *Jenkins*. Neosporivo je da je CI/CD paradigma izuzetno značajna za razvoj IT kompanija sa svim prednostima koje donosi.

Postoji veliki broj alata za implementaciju ove paradigme, pa treba dobro proučiti kada koji iskoristiti, *Jenkins* predstavlja vodeći alat danas sa svim svojim *plugin*-ovima i novim deklarativnim pipeline-om. *Docker* je takođe vodeći alat jer ima podršku za *Kubernetes* koji upravlja kontejnerima na distribuiranim sistemima. Dalji razvoj arhitekture bi uključio *Kubernetes* i distribuciju sistema.

11. LITERATURA

[1] S.A.I.B.S. Arachici, Indika Perera, „Continuous Integration and Continuous Delivery Pipeline Automtion for Agile Software Poject Management“, Department of Computer Science and Engineering, University of Moratuwa. Moratuwa, Sri Lanka

[2] Zebula Sampedro, Aaron Holt, Thomas Hauser, „Continuous Integration and Delivery for HPC: Using Singularity and Jenkins“, Univeristy of Colorado Boulder, Research Computing, Boulder, Colorado

[3] Kim Rejström, „Implementing Continuous Integration in a Small Company: A Case Study“, Aalto University, School of Science

[4] https://jenkins.io/doc/ (pristupljeno u septembru 2019.)

[5] https://docs.docker.com (pristupljeno u septembru 2019.)

Kratka biografija:

Milena Počuča, rođena je 27.08.1994. u Kraljevu. 2013. godine upisuje „Fakultet tehničkih nauka” u Novom Sadu, smer „Računarstvo i automatika”. 2017. godine dobija zvanje Diplomirani inženjer elektrotehnike i računarstva. 2017/18. godine upisuje master akademske studije, smer „Primenjene računarske nauke i informatika – Elektronsko poslovanje”. 01.03.2018. godine dobija zvanje Saradnika u nastavi na Departmanu za računarstvo i automatiku na Fakultetu tehničkih nauka u Novom Sadu. kontakt: minapocuca@gmail.com