Linearni slijed specijaliziranih modula od razvoja do produkcije Java pozadinskih aplikacija

Buljan, Marko

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Humanities and Social Sciences / Sveučilište Josipa Jurja Strossmayera u Osijeku, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:142:717675

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-15**

FILOZOFSKI FAKULTET
SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

Repository / Repozitorij:

FFOS-repository - Repository of the Faculty of Humanities and Social Sciences Osijek

zir.nsk.hr

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet Osijek

Diplomski dvopredmetni studij nakladništva i informacijske tehnologije

Marko Buljan

Linearni slijed specijaliziranih modula od razvoja do produkcije Java pozadinskih aplikacija

Diplomski rad

Mentor doc. dr. sc. Tomislav Jakopec

Osijek, 2023.

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet Osijek

Odsjek za informacijske znanosti

Diplomski dvopredmetni studij nakladništva i informacijske tehnologije

Marko Buljan

Linearni slijed specijaliziranih modula od razvoja do produkcije Java pozadinskih aplikacija

Diplomski rad

Društvene znanosti, Informacijske i komunikacijske znanosti

Mentor doc. dr. sc. Tomislav Jakopec

Osijek, 2023.

Prilog: Izjava o akademskoj čestitosti i o suglasnosti za javno objavljivanje

Obveza je studenta da donju Izjavu vlastoručno potpiše i umetne kao treću stranicu završnog odnosno diplomskog rada.

IZJAVA

Izjavljujem s punom materijalnom i moralnom odgovornošću da sam ovaj rad samostalno napravio te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu označeni kao citati s napisanim izvorom odakle su preneseni.
Svojim vlastoručnim potpisom potvrđujem da sam suglasan da Filozofski fakultet Osijek trajno pohrani i javno objavi ovaj moj rad u internetskoj bazi završnih i diplomskih radova knjižnice Filozofskog fakulteta Osijek, knjižnice Sveučilišta Josipa Jurja Strossmayera u Osijeku i Nacionalne i sveučilišne knjižnice u Zagrebu.

U Osijeku, datum 05.09.2023.

Marko Buljan, 0122228831
ime i prezime studenta, JMBAG

Zahvala

Želio bih se zahvaliti svom stručnom mentoru mag. Stijepi Vrdoljaku na predanosti, pomoći i potpunoj nesebičnoj angažiranosti prilikom izrade ovog diplomskog rada. Konkretno, želim se zahvaliti na pomoći prilikom pronalaska relevantnih izvora informacija za navedenu temu i na pomoći prilikom organizacije samog rada. Isto tako zahvaljujem se i na pomoći prilikom izrade praktičnih prikaza samog rada te konstantnoj dostupnosti za dodatna pitanja. Kao stručni mentor, Vašom stalnom dostupnošću pomogli ste mi u učenju i potpunom razumijevanju koncepata teme te ste time pomogli ne samo meni, nego i svima onima koji će ovaj rad čitati i na tome Vam neizmjerno zahvaljujem.

Sadržaj

1.Uvod ... 1
2. Organizacija rada i upravljanje izvornim kodom ... 3
2.1. Sustav verzioniranja koda ... 4
2.2. Iniciranje Spring backend projekta i repozitorija ... 8
2.2.1. Spring backend projekt ... 9
2.3. Organizacija Kanban metodologijom i izrada Git grana ... 13
3. Kontinuirana metodologija ... 17
3.1. Linearni slijed događaja ... 19
3.2. Gitlab Runner i Runner Executor ... 21
3.2.1. Instalacija i konfiguracija Gitlab Runner-a ... 23
3.3. Definiranje Gitlab zadataka i varijabli u linearnom slijedu događaja ... 30
3.3.1. Opcije grupiranja i pokretanja Gitlab zadataka u linearnom slijedu događaja ... 35
3.4. Tipovi linearnog slijeda događaja prema arhitekturi i opcije pokretanja linearnog slijeda događaja ... 40
4. Principi kontinuirane integracije i kontinuiranog postavljanja u konfiguraciji linearnog slijeda događaja ... 43
5. Zaključak ... 48
5. Literatura ... 49

Sažetak

Kontinuirana metodologija pojam je koji okuplja kontinuiranu integraciju i kontinuirano postavljanje. Principi kontinuirane integracije i kontinuiranog postavljanja odnose se na automatizaciju izgradnje i testiranja koda te dostavljanja koda do krajnjeg korisnika. Njihovi principi jednostavno se mogu ukomponirati u linearni slijed događaja, koji se u kontekstu kontinuirane metodologije, odnosi na prijenos konstrukcije koda kroz različite faze sve do isporuke same aplikacije. Faze u linearnom slijedu događaja uz provjeru može li kod izgraditi aplikaciju i uspijeva li proći testove mogu služiti za definiranje i svih ostalih standarda koje želimo da napisani kod aplikacije postiže. Navedeni standardi mogu se definirati u nekoliko različitih alata poput Jenkins-a, CircleCI-a, Bamboo-a te ostalih, a u samom radu korišten je alat Gitlab CI/CD čije su mogućnosti definiranja linearnog slijeda događaja uz metodologiju rada i sustav verzioniranja koda, prikazane na pozadinskoj aplikaciji napisanoj u Spring boot-u radnom okviru jezika Java-e.

Ključne riječi

Linearni slijed događaja, kontinuirana metodologija, sustav verzioniranja koda, Java, Spring, Gitlab CI/CD, metodologija rada

Repozitorij izvornog koda projekta: https://gitlab.com/TMEMark/master-thesis-project

1.Uvod

Napretkom znanstvenih područja bliskim informacijskoj tehnologiji omogućila se brža i lakša izrada novih i profitnih tehnologija vezanih za komunikaciju. Isplativijom primjenom različitih segmenata potrebnim za postavljanje sustava kojeg poznajemo kao Internet, njegova rasprostranjenost u gotovo svim dijelovima država zapadnog svijeta uvelike je utjecala na svakodnevni život svih ljudi. Primjerice u Europi prema Eurostat-u, statističkom uredu Europske unije, 71% površine unije u 2021. godini imalo je internetsku pokrivenost.[1] Zbog takve rasprostranjenosti razmjena podataka nikada u povijesti nije bila brža, efikasnija i jednostavnija, neovisno o njezinoj svrsi. Različite značajke na mobilnim uređajima, računalima i svim ostalim multimedijskim proizvodima omogućuju ljudima gotovo potpuno konzumiranje i ovisno o kontekstu i dodatno manipuliranje podacima potpuno slobodno, neovisno o privatnoj ili poslovnoj sferi života. Iako smo u 21. stoljeću potpuno naviknuti na razmjenu podataka putem interneta i očekujemo njegovu stalnu dostupnost, internet, jednostavnom definicijom, kao svjetski sustav povezanih računala, svoj početak je imao kao ideja profesora MIT-a Joseph Carl Robnett Licklider-a opisana pod nazivom Galaktička mreža (eng. *Galactic Network*).[2] Njegova ideja više manje je opisivala internet onakvim kakav je i danas te se iz nje internet dalje samo nastavio eksponencijalno razvijati, prvotno kao ARPANET(eng. *Advanced Research Projects Agency Network*), testni program vojske Sjedinjenih Američkih Država koji je prvi primjer komunikacije udaljenih računala, sve do uspostave standardnog protokola za komunikaciju računala TCP(eng. *Transfer Control Protocol*), e-maila i WWW (eng.*World Wide Web*). [3]Nakon sve veće popularizacije i upotrebe interneta i grafičkog korisničkog sučelja u računalima, prve mrežne stranice, izrađene 90-ih godina prošloga stoljeća, na internetu imale su svrhu statičkog prikazivanja podataka.[4] Danas, kako je internet, već prikazanim podacima, sve rasprostranjeniji i sve više se oslanjamo na mogućnosti tehnologije putem web-a, izgradnja web aplikacija zahtjeva od osoba

[1] Usp. Increase in high-speed internet coverage in 2021. URL: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20220822-1#:~:text=In%202021%2C%2070%25%20of%20EU,to%2037%25%20across%20the%20EU. (2023-10-05)

[2] Usp. Leiner, B.M., Cerf, V.G., Clark, D.D., et al. A Brief History of the Internet. ACM SIGCOMM Computer Communication Review, 39, 22-31 str. URL: https://doi.org/10.1145/1629607.1629613 (2023-10-05)

[3] Usp. Isto (2023-10-05)

[4] Usp. Cloud from A to Z: Why static websites?. URL: https://acenet-arc.github.io/cloud_from_a_to_z/why-static-websites/ (2023-10-05)

koje se bave izradom sve veći standard kvalitete i mogućnosti na aplikaciji. Kako bi kvaliteta aplikacije zadovoljila korisnike i same funkcije aplikacije ispunile njihove potrebe postoje tehnologije i prakse koje mogu pomoći osobama koje sudjeluju u procesu izgradnje i održavanju aplikacija. Usluge računalstva u oblaku (eng. *Cloud service*) jedni su od najboljih primjera brzorastućih mogućnosti koje su dostupne korisnicima na internetu i olakšavaju izgradnju aplikacije ili ovisno o usluzi same i jesu aplikacija.[5] IaaS(eng. *Infrastructure as a service*), PaaS (*eng. Platform as a Service*), SaaS (eng. *Software as a Service*) i FaaS (eng. Function-as-a-Service) primjeri su usluga računarstva u oblaku koje uz plaćanje nude pogodnosti, poput iznajmljivanja infrastrukture, potpunog održavanja pozadinskog dijela aplikacije i cjelokupne aplikacije, ali ono što im je zajedničko je olakšati proces izrade same aplikacije. Iako pojedine usluge nude više nego druge sve navedene usluge programerima i svim ostalim sudionicima u procesu izgradnje i održavanja aplikacije omogućuju više vremena za rad s klijentima i za pisanje koda kako bi aplikacija bila što kvalitetnija.[6] No, više vremena za pisanje koda ne osigurava nužno i samu kvalitetu koda. Kako bi kvaliteta koda aplikacije bila na standardiziranoj razini važno je kontrolirati i sam kod. Prakse tj. pristupi radu koji omogućuju i koji će se u ostatku diplomskog rada i prikazivati su kontinuirana integracija i kontinuirano postavljanja aplikacije. Navedene prakse primjenjuju se prilikom izrade dijela aplikacije na strani korisnika (eng. front-end) i prilikom izrade aplikacije na strani poslužitelja (eng. back-end). Primjenjuju se zapravo prilikom izrade bilo kojeg projekta u sferi IT-a(eng. Information Technology) čije se datoteke mogu verzionirati. Stoga s ciljem prikaza svih aspekta praksi kontinuirane integracije i postavljanja aplikacije, u radu je, temeljem back-end dijela aplikacije pisanog u Spring Boot radnom okviru jezika Jave, prikazan rad s kontinuiranom integracijom i kontinuiranim postavljanjem aplikacije. Glavni dio rada podijeljen je na dva dijela. Prvi dio rada odnosi se na organizaciju rada i upravljanje izvornim kodom, koje je ključno za razumijevanje kontinuirane integracije i njezine implementacije u sam proces izrade aplikacije. Drugi dio detaljnije teorijski i praktično opisuje kontinuiranu integraciju i kontinuirano postavljanje aplikacije. Prikazana je definicija cjevovoda(eng. *pipeline*) u IT-u, kontinuirana metodologija te mogućnosti konfiguracije principa kontinuirane metodologije uz pomoć Gitlab CI/CD alata.

[5] Usp. IaaS vs. PaaS vs. SaaS. URL: https://www.ibm.com/topics/iaas-paas-saas (2023-05-08)

[6] Usp. Isto (2023-05-08)

2. Organizacija rada i upravljanje izvornim kodom

Kontinuirana integracija i kontinuirano postavljanje, kako je i ranije navedeno, dvije su prakse tj. dva su pristupa rada na projektu. No, kako bi razumjeli principe za provedbe projekta pomoću CI (eng. *Continuous Integration*) i CD (eng. *Continuous Deployment/Delivery*) važno je razumjeti metodologije rada, odnosno organizaciju rada prilikom izrade aplikacije i što je to sustav za verzioniranje koda. Budući da je organizacija rada i raspodjela obveza timovima, uz što bolju efikasnost, problem s kojim su se voditelji projekata neovisno o tipu posla suočavali kroz cijelu povijest, pokušajem stvaranja što bolje organizacije nastao je neizmjeran broj smjernica, pravila i procesa navođenja koji zajedno čine metodologiju. Ona kao pojam proizlazi iz znanstvene zajednice i prema Hrvatskoj enciklopediji odnosi se ukratko na disciplinu koja proučava logičke operacije i tehničko-istraživačke postupke određene znanstvene grane.[7] Kada se navedeno kontekstualizira u praksu i projekte vezane za industrijske grane, metodologija se može definirati kao skup operacija i pravila koji nam služe za organizaciju i vođenje projekata. Naravno definicija se može i proširiti i iskoristiti i za upravljanje samom firmom i korištenje van informacijske tehnologije, ali za potrebe prikazivanja metodologija u IT-u navedeno je nepotrebno.[8] Kako su se s vremenom zapisivale smjernice i samim radom uviđale dobre prakse nastalo su metodologije rada koje pomažu u vođenju projekata. Prilikom vođenja izgradnje programske podrške, mobilne aplikacije, web aplikacije ili bilo kojeg drugog oblika projekta srodnog IT-u neke od popularnih metodologija prema izvoru SpaceO Technologies u 2023. godini su: Agilna metodologija, DevOps metodologija i Vodopadna metodologija. Naravno sve navedene metodologije nisu jedine popularne metodologije u 2023. godini, ali za potrebe shvaćanja principa rada na projektima u industriji i kako se implementira praksa CI/CD-a s metodologijama rada dovoljno je objasniti navedene.[9] Agilna metodologija za izradu aplikacija kako i sam naziv nalaže dopušta prilagodljivost zahtjevima projekta i različitim utjecajima koji mogu utjecati na tijek provedbe projekta. Navedeno se postiže dijeljenjem posla u manje vremenske segmente koji se nazivaju Sprintevi i stalnom komunikacijom s klijentima. Za vrijeme svakog Sprinta obavljaju se dodijeljeni zadaci te se na kraju svakoga uviđa koji su zadaci idući na prioritetnoj listi prema razgovoru s

[7] Usp. Metodologija. URL: https://www.enciklopedija.hr/natuknica.aspx?id=40441 (2023-05-08)

[8] Usp. Isto (2023-05-08)

[9] Usp. 11 Top Most Popular Software Development Methodologies. URL: https://www.spaceo.ca/blog/software-development-methodologies/ (2023-05-10)

klijentima.[10] Nadalje DevOps metodologija nije samo metodologija za izradu aplikacija već set aktivnosti za održavanje cijele organizacijske kulture. Temelji se na organizaciji rada u više radnih segmenata koji se ponavljaju. Neki od radnih segmenata su: planiranje, kodiranje, testiranje, postavljanje, održavanje.[11] Zadnja spomenuta metodologija, Vodopadna metodologija, temelji se na linearnom slijedu ciklusa. Jednostavna je za praćenje jer ima strogo definirano slijedno izvođenje aktivnosti, ali za razliku od agilne metodologije nije prilagodljiva potencijalnim promjenama u procesu izgradnje aplikacije. Sve navedene metodologije, budući da se koriste za izradu proizvoda u IT industriji imaju zajedničke aspekte rada drugačije organizirane. U Agilnoj metodologiji dijelovi aplikacije izrađuju se u Sprintevima prema zahtjevima korisnika, u Vodopadnoj metodologiji isti ti zahtjevi rješavaju se jedan po jedan, a u DevOps metodologiji prati se slijed radnih procesa. Ono što je bitno je da na svakom zahtjevu korisnika neovisno o metodologiji i organizaciji rada, radi više ljudi.[12] S obzirom na suradnju više ljudi na svakom zahtjevu te niz faza kroz koje prolazi svaki zahtjev, koje obuhvaćaju: dizajniranje, planiranje, kodiranje, testiranje i objavljivanje, sam proces izrade iznimno je kompleksan. Stoga, ključno je implementirati sustav za verzioniranje koda i prakse kontinuirane integracije i kontinuiranog postavljanja kako bi se izbjegla redundantnost poslova i osigurala dosljedna primjena kvalitete koda u skladu s metodologijom.

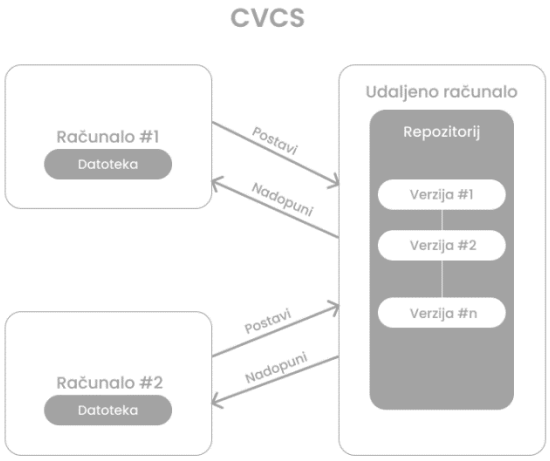
2.1. Sustav verzioniranja koda

Sustav verzioniranja koda je sustav koji služi za upravljanje procesom izrade određenog koda, odnosno sustav koji bilježi promjene na kodu. On ljudima koji rade na kodu omogućuje spremanje koda na zajednički repozitorij te time svakom programeru koji sudjeluje u projektu osigurava posljednju verziju koda dostupnom.[13]Sam kod koji se objavljuje na zajednički repozitorij može biti podijeljen na više segmenata. U sustavu verzioniranja koda postoji mogućnost grananja gdje se najčešće, iz organizacijske perspektive posla, iz glavne verzije koda sistematiziraju podgrane na kojima se aktivno radi ovisno o zahtjevu koji je dan programerima.[14] Navedeni zahtjevi mogu biti problemi nastali na aplikaciji, zahtjevi od korisnika, arhitektonske promjene i bilo koji drugi

[10] Usp. Isto (2023-05-10)
[11] Usp. Isto (2023-05-10)
[12] Usp. Isto (2023-12-05)
[13] Usp. 1.1 Getting Started - About Version Control. URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (2023-05-12)

[14] Usp. Isto (2023-05-12)

zahtjevi koji mogu pridonijeti u izradi kvalitetne aplikacije. Oni se zapisuju i vode prema metodologiji rada i radi organizacije nakon završavanja koda za određeni zahtjev kod se dodaje sa sporedne grane u glavnu granu.[15] Tim načinom osim dostupnosti zadnje verzije koda svim sudionicima na projektu, osigurava se i vođenje projekta prema organizacijskog metodologiji gdje se svaka sporedna grana naziva prema opisu zahtjeva te da je sam kod na sporednoj grani uvijek provjeren prije spajanja na glavnu granu kako bi osigurali kvalitetan i valjan kod.[16] Budući da se ideja verzioniranja koda razvijala kroz povijest nastala su i 2 tipa sustava za verzioniranje koda s vlastitim dijelovima i funkcijom. Prvi, ujedno i stariji oblik sustava za verzioniranje koda, CVCS (eng. *Centrailzed Version Control Systems*), kako mu i sam naziv nalaže, centralizirani je sustav koji ima jedan repozitorij na udaljenom računalu.[17] Dopušta programerima zajednički rad tako što pohranjuje sve promjene na kodu i datotekama na repozitorij kojima programeri mogu pristupiti spajanjem na udaljeno računalo na kojem se repozitorij nalazi. [18]



Slika 1. *Prikaz sheme CVCS-a*

[15] Usp. Gitflow-Workflow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (2023-12-05)

[16] Usp.Isto (2023-05-12)

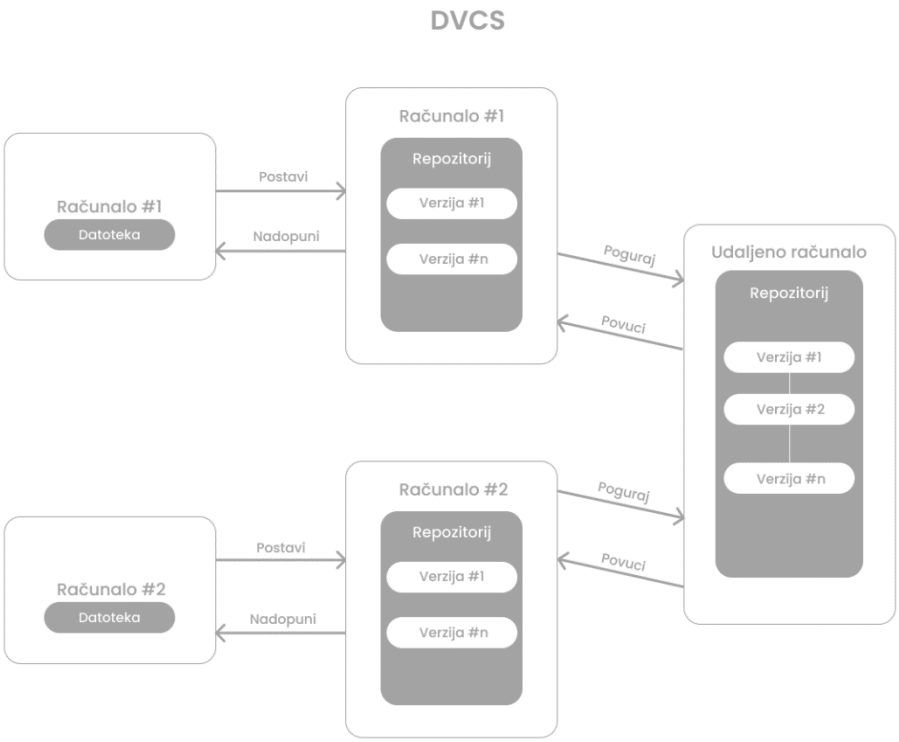
[17] Usp. 1.1 Getting Started - About Version Control. URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[18] Usp. Isto (2023-12-05)

Kao što se može vidjeti na slici 1 CVCS omogućuje programerima nakon spajanja na udaljeno računalo s repozitorijem povlačenje posljednjih promjena s repozitorija naredbom *pull* i postavljanje promjena u kodu i općenito datotekama naredbom *push*. Budući da su sve posljednje promjene spremljene na udaljeno računalo pod najnovijom verzijom, sva računala spojena imaju pristup posljednjim promjenama na datotekama. Osim povlačenja i postavljanja promjena važna operacija prilikom rada sa VCS-om je i već navedeno grananje.[19] Svaki repozitorij CVCS ima jednu glavnu granu koja se prema temeljnim postavkama naziva *main branch* iako sami korisnici mogu i stavit vlastito ime glavne grane na koju se postavlja kod. CVCS omogućuje i već opisano grananje stvaranjem sporednih grana iz glavne grane i po potrebi stvaranjem dodatnih grana iz sporednih grana budući da je praksa povlačenja promjena, grananja postavljanja posljednjih promjena na granu i spajanje sporedne grane s glavnom granom u općenitom kontekstu temelj rada u većini IT firmi.[20] No, iako CVCS omogućuje grananje i rad na zajedničkom repozitoriju sve je manje popularan zbog činjenice da se verzije koda pohranjuju samo na udaljeno računalo i programeri koji rade s kodom imaju dostupnu samo posljednju verziju koda što može biti problematično u slučajevima potrebe vraćanja na stariju verziju ili jednostavno potrebe za osiguravanjem dostupnosti koda na više mjesta od samog udaljenog računala.[21] Kako bi se navedeni problem riješio osmišljeno je rješenje u kojem svaki programer koji radi na kodu uz repozitorij na udaljenom računalu ima i lokalni repozitorij za verzioniranje koda na kojem se bilježe promjene. Navedeni sustav nazvan je DVCS(eng. *Distributed Version Control Systems*). Ono što DVCS omogućuje programerima i svim ostalim sudionicima u izradi projekta je pohranjivanje svih verzija datoteka na repozitorij na lokalnom računalu i na udaljenom računalo.[22] Time timovi mogu komunicirati međusobno i održavati vlastite lokalne repozitorije bez potrebe da su stalno spojeni na repozitorij na serveru. Isto tako, najvažnija prednost DVCS-a naspram CVCS-a je kreiranje dodatne sigurnosne kopije verzija svih datoteka na lokalnim repozitorijima kod svakog projektnog sudionika što omogućuje veću sigurnost datoteka i podataka. [23]

[19]Usp. Version Control System. URL: https://www.geeksforgeeks.org/version-control-systems/ (2023-05-12)

[20] Usp. Isto (2023-05-12)
[21] Usp. Isto (2023-05-12)
[22] Usp. Isto (2023-05-12)
[23] Usp. Isto (2023-05-12)



Slika 2. *Prikaz sheme CVCS-a*

Kao što se može vidjeti na 2. slici DVCS funkcionira slično kao i CVCS s razlikom u tome što programeri koji rade imaju i lokalni repozitorij na svom računalu na koji prvo postavljaju promjene naredbom *commit,* a sve promjene koje nisu poslane na repozitorij na serveru postavljaju naredbom *push*. Što se tiče povlačenja posljednjih promjena na serveru koristimo se kao i ranije narednom *pull* koja povlači promjene na lokalni repozitorij, a nakon toga naredbom *update* za povlačenja promjena s repozitorija s lokalnog repozitorija.[24] No, iako DVCS i CVCS osiguravaju spremanje promjena na datotekama u zajednički repozitorij, grananje prema potrebama zahtjeva koji prate metodologiju rada i što se CVCS-a tiče i lokalni repozitorij ono što ne omogućavaju je osiguravanje valjanosti koda i provjeravanje aplikacija. Što ima smisla budući da je svrha VSC-a, neovisno o tipu, verzioniranje koda i svih datoteka vezanih za projekt, a ne i potvrđivanje da sam kod doista i radi prema standardima koje pojedini projekt očekuje.[25] Stoga za definiranje pravila i pojedinosti koje pojedini kod i datoteke projekta moraju ispoštovati prilikom svakog postavljanja

[24] Usp. Isto (2023-05-12)
[25] Usp. Isto (2023-05-12)

koda na granu repozitorija na serveru, tj. *push-a* ili spajanja grane s drugom granom na repozitoriju *merge-a* potrebno je implementirati principe kontinuirane integracije, a za automatizaciju objavljivanja aplikacije praksu kontinuiranog postavljanja čija je praktična i teorijska implementacija detaljnije opisana u 3. poglavlju temeljem Spring backend projekta.[26]

2.2. Iniciranje Spring backend projekta i repozitorija

Spring je radni okvir otvorenog pristupa za Java programski jezik.[27] Java je nastala 1995. godine kao objektno orijentirani jezik s kompajlerom koji prevodi izvorni kod u kod razumljiv računalu odnosno byte.[28] On se izvodi u specifikaciji koja omogućuje pokretanje koda neovisno o radnom okruženju te se naziva JVM(eng. *Java Virtual Machine*). Ono što je važno razumjeti je da specifikacija za izvođenje koda je različita od platforme do platforme i sama specifikacija ovisi o platformi, ali sam Java *kod* je potpuno neovisan jer se sam *kod* prije izvođenja prvo uvijek kompajlira.[29] No, osim vlastitog koda koji napišemo, u Javi postoje i različite biblioteke koda s definiranim funkcijama koje ne postoje u samoj Javi kojima možemo pristupiti uz pomoć JRE(eng. *Java Runtime Environment*), a sve navedeno dio je JDK(eng. *Java Development Kit*) s još dodatnim mogućnostima poput vlastitog kompajlera Javac-a, generatora dokumentacije i ostalim mogućnostima.[30] Budući da je Java, kao takva višeplatformska i prema mnogima i sama definirana kao platforma, velik broj IT projekata upravo je napisan u Javi te kako je sve više programera radilo s Javom sami su počeli za vlastite potrebe definirati metode i strukture aplikacije koje se mogu uspostaviti za generalnu i ponovnu upotrebu te su na taj način počeli nastajati radni okviri od kojih je jedan i sam Spring. Inicijalno ga je napisao Rod Johnson, a izdan je pod Apache 2.0. licencom 2003. godine. Spring radno okruženje je platforma koja programerima nudi podršku u vidu strukture.[31] Budući da je Spring radno okruženje za Javu, a kako je Java objektno orijentirani

---

[26] Usp. Isto (2023-05-12)

[27] `Usp. Introduction to the Spring Framework: Part I. Overview of Spring Framework. URL: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html (2023-05-15)

[28] Usp. What is Java ? URL: https://www.javatpoint.com/java-tutorial (2023-05-12)

[29] Usp. Difference between JDK, JRE, and JVM. URL: https://www.javatpoint.com/difference-between-jdk-jre-and-jvm (2023-05-12)

[30] Usp. Isto (2023-05-)

[31]Usp. Spring Framework – Overview. URL: https://www.tutorialspoint.com/spring/spring_overview.htm (2023-12-05)

jezik, sam Spring radi s klasama i objektima i glavni problem koji Spring rješava umjesto programera je upravljanje objektima. Prema temeljnim postavkama Spring-a svaka klasa funkcionira prema Singleton principu. No, Spring singleton princip ne treba miješati s Java-inim Singleton principom.[32] U Java-i Singleton princip odnosi se na činjenicu da se svaka klasa može instancirati samo jedanput, odnosno svaka klasa može imati samo jedan objekt.[33] Dok Spring Singleton princip da bi razumjeli moramo razumjeti kako funkcioniraju koncepti *bean-a* i Spring IoC (eng. *Inversion of Controll*) *container-a*. Spring *bean* prema opisu je svaki objekt koji se inicijalizira u Spring IoC *container-u.* IoC je mehanizam koji služi za povezivanje svih objekata prilikom pokretanja aplikacije, a IoC container je program u Springu koji ubacuje ovisnosti u objekte te na taj način povezuje svaki objekt neovisno o tome dali je objekt nastao iz već postojeće klase u Spring-u ili je instanciran iz klase kreirane od programera.[34] Stoga u Springu Singleton princip podrazumijeva jedan bean prema jednom Spring container-u te se svaki definirani bean dijeli između svih drugih radi povezivanja svih objekata te tim principom Spring upravlja objektima umjesto nas.[35] Osim navedenog Spring omogućuje i jednostavno dodavanje vanjskih biblioteka u aplikaciju uz pomoć IoC *container*-a. Uz sve navedeno prednost Springa je i što svojom arhitekturom je prilagođen izradi web aplikacija, ekonomičan je prilikom zauzimanja prostora memorije računala te je i brz. Zbog svega prikazanog odabran je kao radni okvir na temelju kojeg će se izraditi demonstrativna aplikacija za prikaz mogućnosti CI/CD principa.[36]

2.2.1. Spring backend projekt

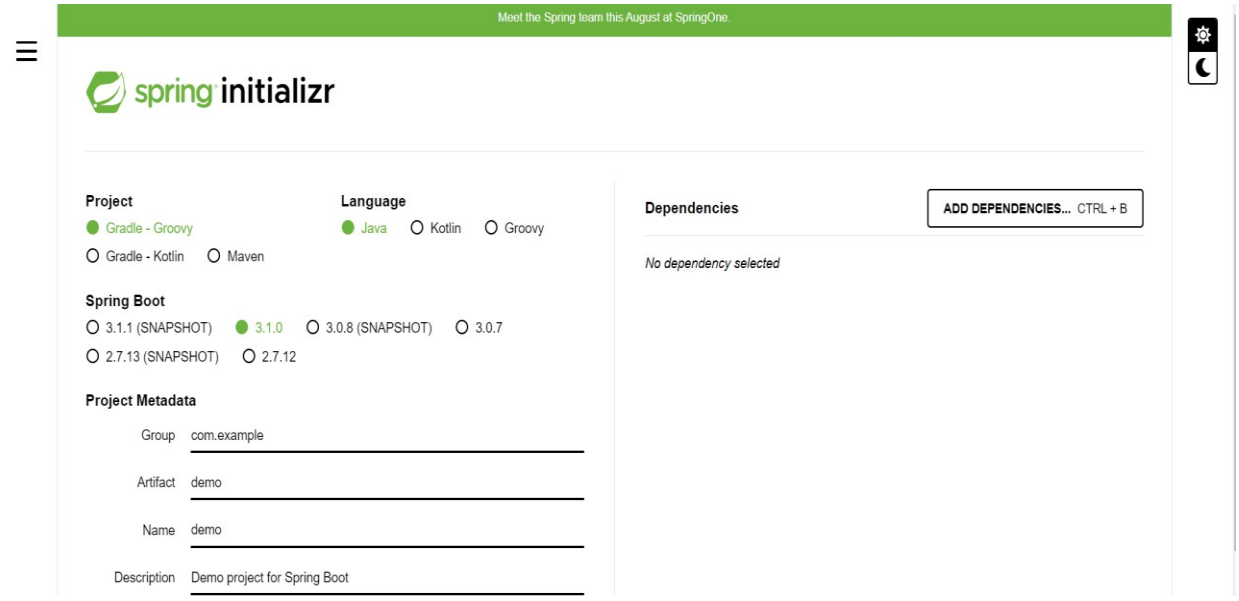
Izrada temeljnog Spring projekta može se provesti kroz web ili u radnom okruženju koje dopušta iniciranje Spring projekta. Na WEB-u se Spring projekt jednostavno kreira korištenjem Spring-ove službene stranice za pokretanje projekta.[37]

---

[32] Usp. Isto (2023-05-13)
[33] Usp. Singleton in Java. URL: https://refactoring.guru/design-patterns/singleton/java/example (2023-13-05)

[34] Usp. Introduction to the Spring Framework: Part I. Overview of Spring Framework. URL: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html (2023-05-13)

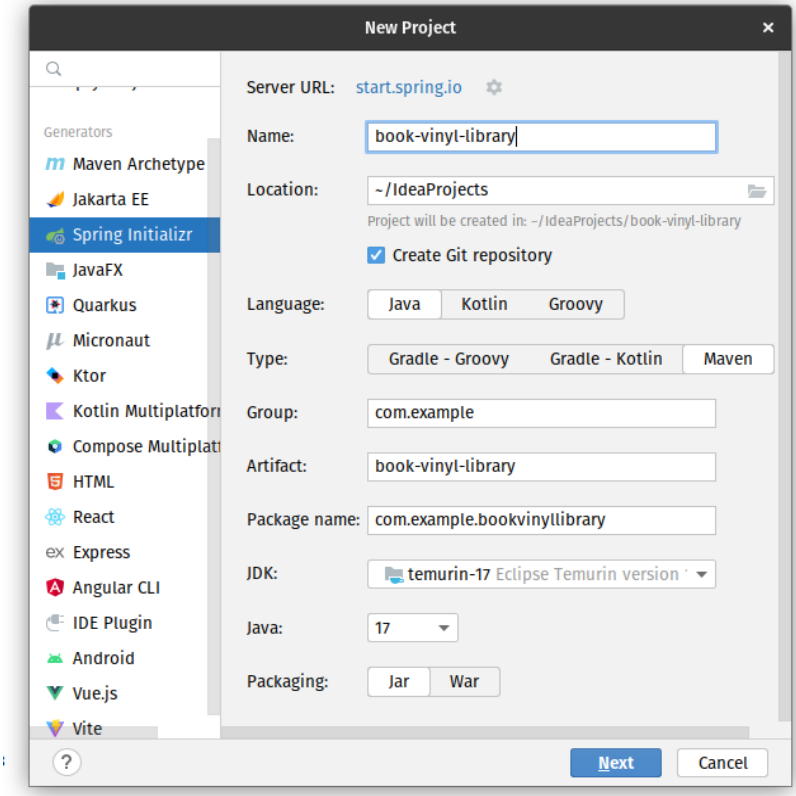
[35] Usp. Isto (2023-05-13)
[36] Usp. Isto (2023-05-13)
[37] Usp. Building an Application with Spring Boot. URL: https://spring.io/guides/gs/spring-boot/ (2023-20-05)



Slika 3. *Prikaz izrade Spring projekta u web sučelju*

Ona omogućuje jednostavno definiranje naziva projekta, za koji jezik je konfiguriran projekt, verziju i odabir programa za upravljanje ovisnostima te ostale mogućnosti. Prikazani način izrade temeljnog Spring projekta na slici 3 koristan je u slučaju kada samo radno okruženje u kojem programer piše kod nema mogućnost izrade Spring projekta te na ovaj način omogućuje se izrada i dodavanje projekta na lokalno radno okruženje programera.[38] Iako je izrada Spring projekta kroz radno okruženje praktična, ukoliko radno okruženje dopušta efikasnije je kreirati sam Spring projekt kroz sučelje radnog okruženja zbog jednostavne činjenice da ga radno okruženje automatski prepoznaje i smješta u mapu na lokalnom računalu s ostalim projektima automatski.[39] IntelliJ IDEA(eng. *Integrated Development Environment*) radno je okruženje JetBrains-a, tvrtke koja proizvodi programe za rad s kodom u različitim jezicima, a IntelliJ je posebno napravljen za rad s Java-om.[40] Budući da je posebno napravljen za rad s Java-om IntelliJ omogućuje izradu Spring projekta poput web okruženja samog Spring-a.[41]

[38] Usp. Isto (2023-05-20)
[39] Usp. Isto (2023-05-20)
[40] Usp. IntelliJ IDEA overview. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (2023-05-20)
[41] Usp. Tutorial: Create your first Spring application. URL: https://www.jetbrains.com/help/idea/your-first-spring-application.html (2023-05-20)

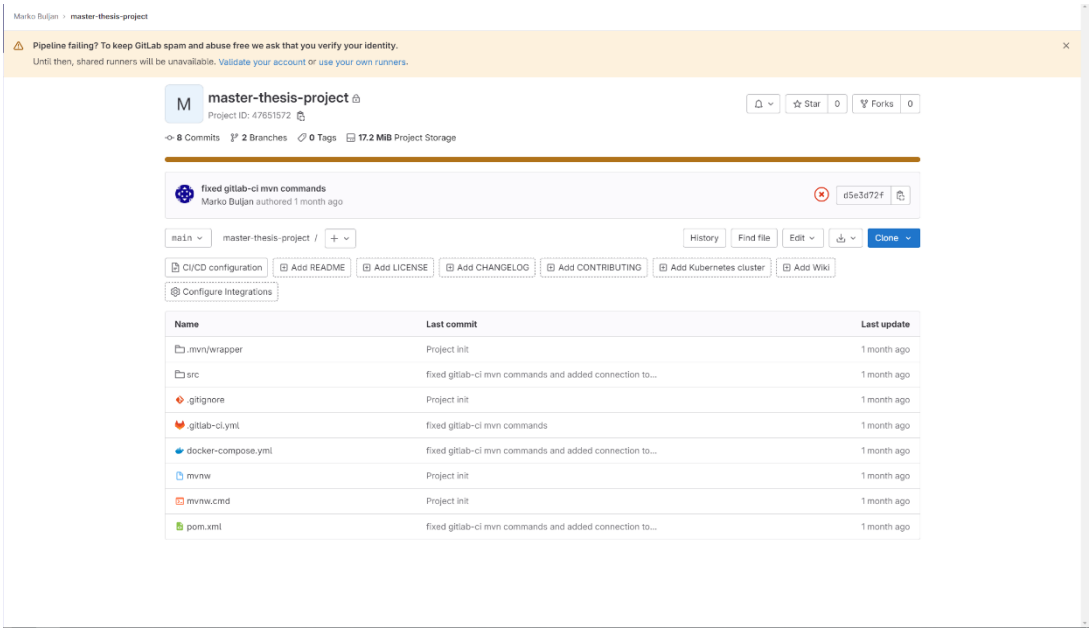


Slika 4. *Prikaz izrade Spring projekta u IntelliJ radnom okruženju*

Uz gotovo iste opcije kao i web inačica izrade Spring projekta, izrada projekta kroz IntelliJ radni okvir nudi još odabir verzije JDK-a i automatsku izradu Git repozitorija što se vidi i na slici 4.[42] Ono što se još može vidjeti je da je naziv projekta *book-vinyl-library*. Kako sam naziv i nalaže, koji je na Engleskom radi olakšavanja zajednici koja radi s projektima u IT-u da pronađe sam repozitorij i iščita ga, projekt na kojem su demonstrirani principi CI/CD-a je jednostavan back-end dio aplikacije za vođenje podataka o knjigama i pločama. Aplikacija korisnicima nudi opcije spremanja ploča ili knjiga, koje oni žele imati ili one koje već imaju, uređivanje vlastitog profila i služi za vođenje podataka o knjigama i pločama. Budući da je aplikacija samo demonstrativne naravi i nije naglasak na prikazivanju njezine izrade već na prikazivanju upravljanja kvalitetom izrade aplikacije i postavljanjem aplikacije na server sama aplikacije nije kompleksna niti pretjerano kreativna, ali služi svojoj svrsi. Nakon inicijalne izrade aplikacije uz njezinu izradu,

[42] Usp. Isto (2023-05-20)

izrađen je i Git lokalni repozitorij za verzioniranje koda. Osim lokalnog repozitorija prije nastavka rada na aplikaciji potrebno je osigurati i repozitorij za verzioniranje koda na udaljenom računalu. Postoji nekoliko platformi koje omogućuju verzioniranje koda poput Github-a, Google Code-a, BitBucket-a, Gitlab-a i drugih.[43] Sve navedene platforme su besplatne i omogućuju korisnicima pretraživanje ostalih repozitorija, postavljanje vlastitih repozitorija za verzioniranje koda, upravljanje profilom te brojne druge opcije koje olakšavaju rad na projektu i osiguravaju dostupnost koda, a za potrebe projekta izabran je Gitlab.[44] Prvi korak potreban za izradu repozitorija na Gitlab-u je naravno otvaranje računa na kojem će se i registrirati repozitorij.[45]



Slika 5. *Prikaz repozitorija na Gitlab-u*

Nakon registracije repozitorija prikazanog na slici 5 potrebno je povezati lokalni repozitorij projekta s udaljenim repozitorijem. Jednostavna naredba *git remote add origin* uz link repozitorija unesena u komandno sučelje projekta dovoljna je za povezivanje repozitorija.

[43] Usp. Best Git Hosting Services. URL: https://nira.com/best-git-hosting-services/
[44] Usp. Isto (2023-05-20)
[45] Usp. Repository. URL: https://docs.gitlab.com/ee/user/project/repository/ (2023-05-20)

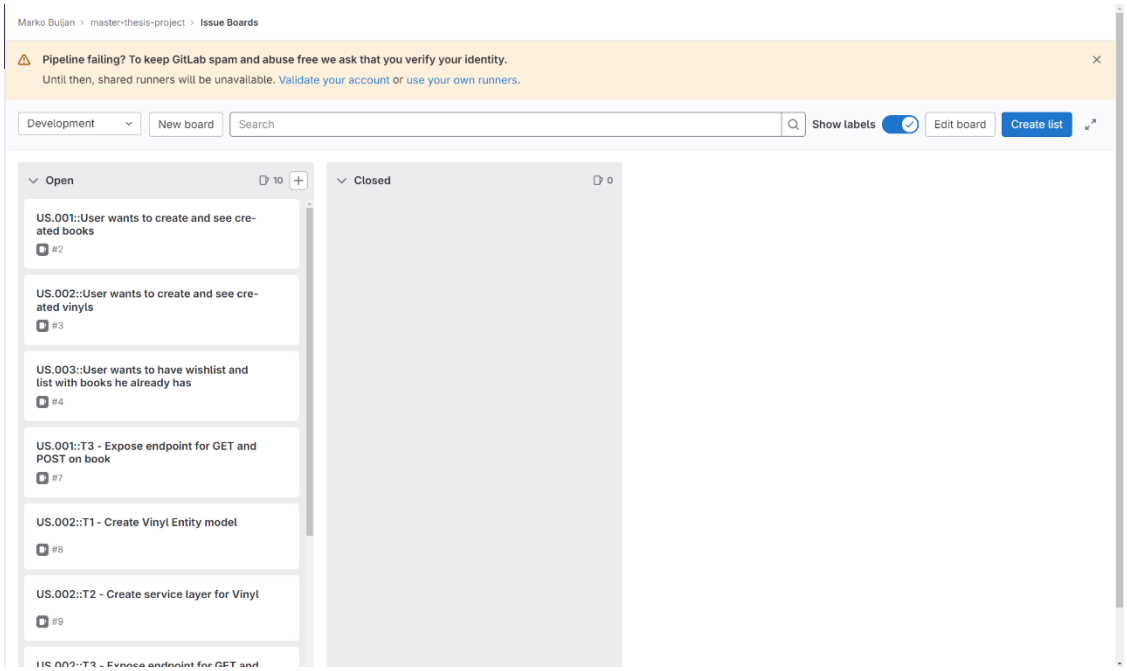
Slika 6. *Prikaz naredbe povezivanja repozitorija na udaljenom računalu s projektom*

Unosom naredbe u terminal dostupan kroz samo radno okruženje prikazano na slici 6 ili kroz komandno sučelje operacijskog sustava povezuje se repozitorij.[46] Nakon povezivanja repozitorija na Gitlab-u s projektom poželjno je uz određenu metodologiju organizacije rada uklopiti zadatke koji se moraju izvesti na aplikaciji kako bi ona ispunjavala zahtjeve korisnika. Budući da Gitlab nudi korisnicima postavljanje zadataka na ploču u kategorije, Kanban metodologija je ona koja se koristila za ukomponiranje zadataka za izvođenje projekta.

2.3. Organizacija Kanban metodologijom i izrada Git grana

Metodologije ili metode rada u praktičnom okruženju, kao što je i ranije navedeno, služe nam za organizaciju i upravljanje provedbom određenog procesa, neovisno o tome koje je proces prirode.[47] U direktnom slučaju izrade aplikacije proces ukomponira sve one aktivnosti koje dovode do krajnjeg proizvoda odnosno završne aplikacije. Kanban metodologijom aktivnosti se organiziraju na ploči od koje i sam naziv metodologije dolazi budući da Kanban na Japanskom jeziku znači ploča. Kako je ona metodologija koja sve aktivnosti čini konstantno dostupnima svima koji sudjeluju u procesu optimalna je za pristup stalnim informacijama o tome koje su se aktivnosti provodile od početka do kraja provedbe procesa.[48] Osim stalnog pristupa informacijama o aktivnostima ona je prvenstveno metoda specifična za razvoj programa u IT-u s naglaskom na organizaciji aktivnosti u kategorije koje su smislene prema tijeku izvođenja samih procesa.[49]

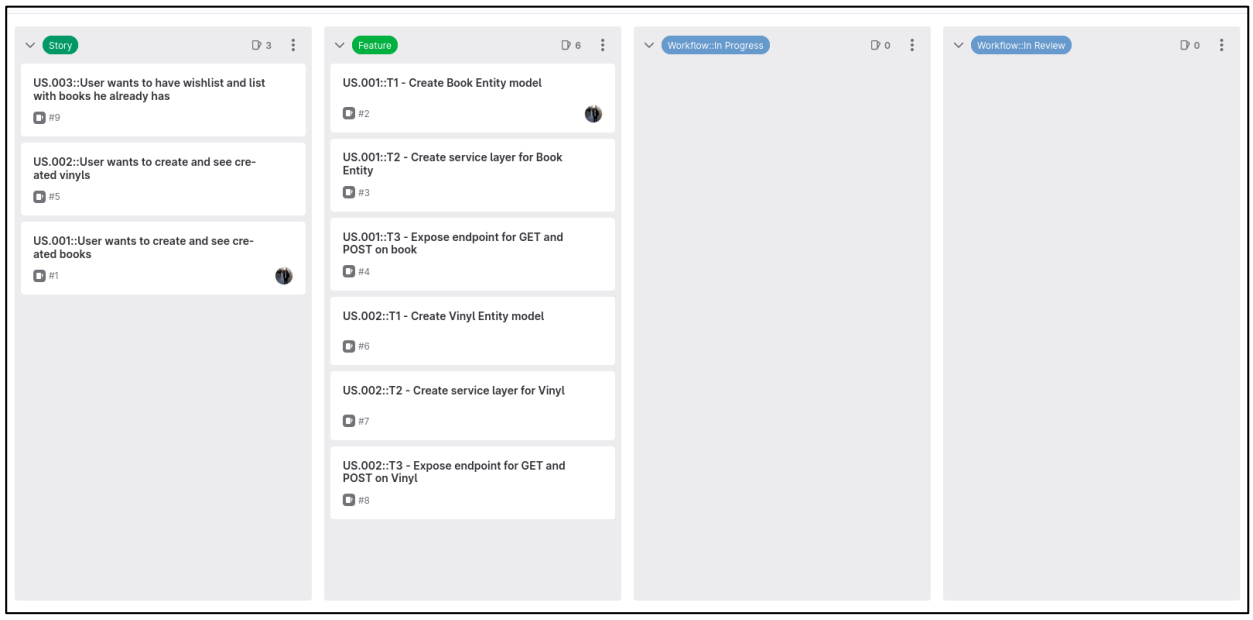
[46] Usp. Isto (2023-05-20)
[47] Usp. Metodologija. URL: https://www.enciklopedija.hr/natuknica.aspx?id=40441 (2023-05-23)
[48] Usp. Kanban – Agile Methodology. URL: https://www.geeksforgeeks.org/kanban-agile-methodology/ (2023-05-23)
[49] Usp. Isto (2023-05-23)



Slika 7. *Prikaz ploče metodologije na Gitlab-u s početnim kategorijama*

Prema temeljnim postavkama Gitlab ploča za organizaciju aktivnosti nudi dvije kategorije: *open,* u koju se postavljaju aktivnosti spremne za rad i *closed,* za aktivnosti koje su ili završene ili je odlučeno da se neće provesti.[50] No, kako je izrada projekta neovisno o grani proizvodnje kompleksna tako su i IT projekti kompleksni te za upravljanje samim projektom potrebne su detaljnije razrađene kategorije. Gitlab za poboljšanje upravljanja projektom, prilagođeno potrebama projekta, nudi dodavanje više lista od početnih kako bi se što eksplicitnije moglo definirati u kojem je stadiju određena aktivnost u procesu izrade aplikacije.[51]

[50] Usp. Issue boards. URL: https://docs.gitlab.com/ee/user/project/issue_board.html (2023-05-24)
[51] Usp. Isto (2023-05-23)

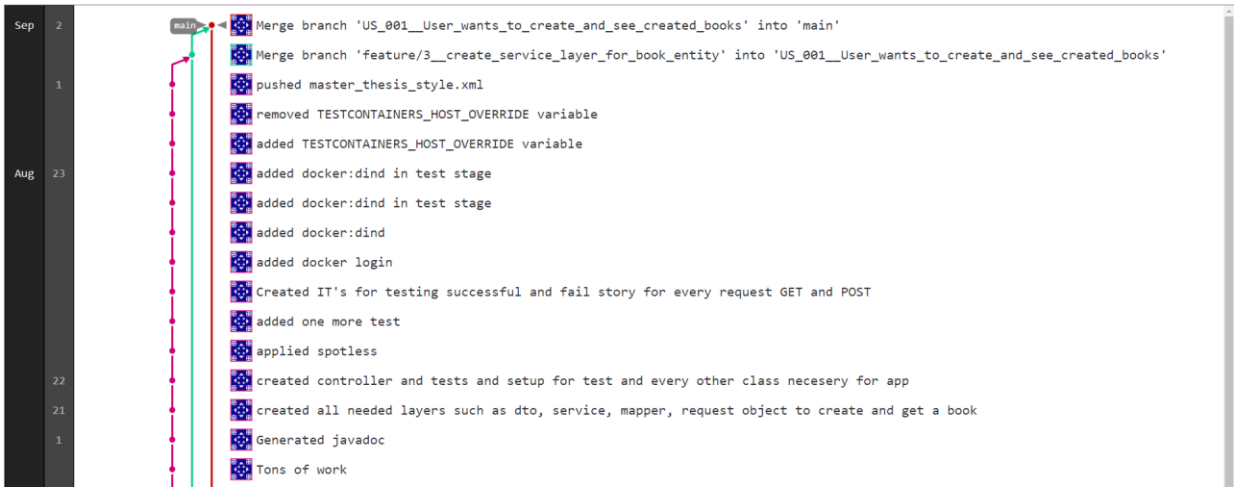


Slika 8. *Prikaz ploče metodologije na Gitlab-u s proširenim kategorijama*

Specifično za potrebe aplikacije *Book vinyl library* definirane su četiri dodatne kategorije. Kategorija *story* u sebi sadržava aktivnosti koje opisuju što korisnik želi i očekuje od aplikacije, primjerice korisnik želi izraditi i vidjeti izrađene knjige. Prema aktivnostima u *story* kategoriji izrađeni su zadaci u kategoriji *feature*, kako bi se svaka aktivnost u listi *story* provela.[52] Za provedbu svake aktivnosti u *story* kategoriji potrebno je za pojedinačnu aktivnost definirati više zadataka za njezinu provedbu unutar kategorije *feature* kako bi se aktivnost i provela. Nakon što su definirani zadaci svaki programer koji radi na projektu uzima pojedinačni zadatak i stavlja ga u kategoriju *workflow:in progress* te nakon što je gotov stavlja kod koji ispunjava zahtjeve zadatka na provjeru te sam zadatak na ploči u kategoriju *workflow:in review*. Sve navedene kategorije imaju svrhu bolje organizacije i upravljanja aktivnostima u procesu izgradnje aplikacije kako bi sam proces imao što manje grešaka i kako se zadaci na kojima se radi ne bi ponavljali radi što veće isplativosti.[53] No, kako proces izrade aplikacije nije jednodimenzionalan proces u kojem se samo razvijaju mogućnosti bez grešaka, Gitlab konfiguracija Kanban metodologije nudi i izradu više ploča za praćenje aktivnosti te prema tome poželjno je izraditi i ploču s kategorijama za vođenje primijećenih problema na aplikaciji te njihovo popravljanje, ali i bilo koju drugu ploču koja

[52] Usp. Isto (2023-05-23)
[53] Usp. Isto (2023-05-23)

olakšava organizaciju.[54] Iz perspektive organizacije definiranjem više detaljnije opisanih kategorija Kanban metodologijom i ozbiljnošću ljudi koji sudjeluju u projektu da se i drže same organizacije postiže se učinkovitija izvedba projekta gdje se štedi na resursima što je imperativ uz postizanje cilja svakog projekta. Kako je što učinkovitije postizanje cilja važan aspekt svakog projekta iz ekonomske perspektive organizacija aktivnosti Kanban metodologijom može se iskoristiti i za organizaciju grana u sustavu za verzioniranje koda radi povezivanja aktivnosti s ploče i grana u Git-u.[55] Prema tome nakon iniciranja projekta i definiranja svih aktivnosti svaki programer nakon odabira aktivnosti koju izvodi prema njoj u svom lokalnom repozitoriju projekta odvaja granu prvo *story* aktivnosti, a naknadno zadatka iz *feature* kategorije koji ispunjava *story* aktivnost.[56]



Slika 9. *Pojednostavljeni grafički prikaz Git grananja*

Navedenim se postiže organizacijska povezanost aktivnosti postavljenih Kanban metodologijom te se olakšava upravljanje zadacima i olakšava povezivanje potrebe napisanog koda sa zadatkom definiranim na ploči i osobi koja pregledava kod jednostavniji pregled svrhe koda. Osim organizacije i upravljanja aktivnostima važan segment prilikom provedbe projekta je i ispunjavanje korisničkog zadovoljstva stvaranjem proizvoda koji je prilagođen korisničkom iskustvu. Ispunjavanje korisničkog zadovoljstva pokriva se radom na više dijelova aplikacije, ali važan dio je izrada aplikacije pisanjem koda koji ne izaziva greške prilikom korištenja aplikacije.

[54] Usp. Isto (2023-05-23)
[55] Usp. Gitflow-Workflow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
[56] Usp. Isto (2023-05-23)

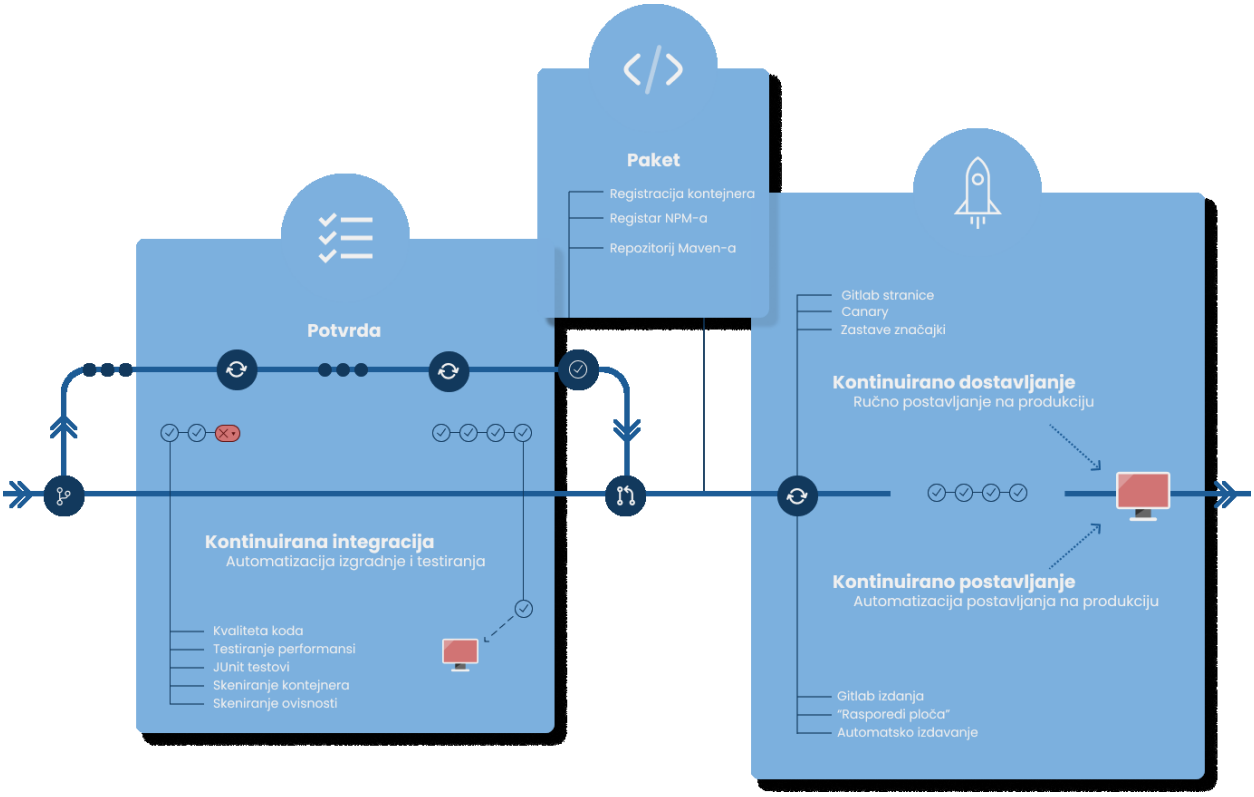
Stoga kako bi se iz ekonomske perspektive postigla ušteda resursa i što veći profit te iz društvene perspektive ugled, osim organizacijom i upravljanjem aktivnosti na projektu važno je postaviti i standarde koje aktivnosti koje se provode moraju ispuniti, odnosno u slučaju IT projekta definiranje standarda koji kod određene aktivnosti mora postići. Navedeni standardi se definiraju dogovorom, a njihova implementacija i automatsko provođenje nad kodom prilikom svakog postavljanja koda na repozitorij provodi se uz pomoć kontinuirane integracije.

3. Kontinuirana metodologija

Kontinuirana metodologija pojam je koji okuplja kontinuiranu integraciju i kontinuirano postavljanje. Definicijom Red hat-a kontinuirana metodologija služi za stalno postavljanje aplikacije krajnjim korisnicima automatizacijom dijelova izrade aplikacije.[57] Budući da su i kontinuirana integracija i kontinuirano postavljanje i isporuka dio kontinuirane metodologije, kako bi se detaljnije shvatili koncepti kontinuirane metodologije potrebno je detaljnije opisati sam pojam što je on, koja mu je svrha i kako se provodi. Kako bi se navedeno postiglo potrebno je analizirati dijelove kontinuirane metodologije i prvotno njih definirati. Kontinuirana integracija praksa je stalne integracije koda. Obuhvaća set skripti koje služe za automatizaciju izgradnje i testiranja koda prije integracije koda na repozitorij.[58] Time kontinuirana integracija provjerava uspijeva li se s napisanim kodom aplikacija izgraditi i prolazi li testove, odnosno standardne zahtjeve kvalitete koja se očekuje od napisanog koda. Ono što ona omogućuje je ušteda vremena prilikom provjere uspijeva li kod postavljen na repozitorij pokrenuti aplikaciju i prolazi li testove.[59] Nadalje kontinuirana isporuka praksa je stalnog dostavljanja promjena na kodu u produkciju odnosno krajnjim korisnicima. Prije samog izdavanja na produkciju kod se provjerava u kontinuiranoj integraciji te tek kad se automatizacijom utvrdi da napisani kod prolazi testove i uspijeva podići aplikaciju može biti isporučen krajnjim korisnicima na produkciji.[60] Kontinuirana isporuka odvija se za razliku od kontinuirane integracije potpuno ručno čime se ostavlja mogućnost dodatne provjere koda prije samog slanja do krajnjeg korisnika. Budući da se kontinuirana isporuka provodi ručno samim time u sukobu je s ranije navedenom definicijom kontinuirane

[57] Usp. What is CI/CD. U https://www.redhat.com/en/topics/devops/what-is-ci-cd (2023-06-15)
[58] Usp. CI/CD Concepts. URL: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration (2023-06-15)
[59] Usp. Isto (2023-06-15)
[60] Usp. Isto (2023-06-15)

metodologije. Osim kontinuirane isporuke postoji još i praksa kontinuiranog postavljanja.[61] Razlika između kontinuiranog postavljanja i kontinuirane isporuke je u tome što se korištenjem prakse kontinuiranog postavljanja cjelokupni proces automatizira. Time se omogućuje potpuna automatizacija isporučivanja aplikacije do krajnjeg korisnika, ali naravno samo onda kada *kod* aplikacije uspješno izgradi aplikaciju i uspješno prođe sve zahtjeve napisane u kontinuiranoj integraciji.[62]



Slika 10. *Grafički prikaz procesa kontinuirane metodologije*

Cjelokupni proces provođenja svih segmenata kontinuirane metodologije vidljiv je na slici 10 od kontinuirane integracije do kontinuirane isporuke ili postavljanja. Iz navedenoga zaključuje se jednostavno da je Kontinuirana metodologija pojam koji objedinjuje praksu kontinuirane integracije, za potrebe provjere valjanosti koda s praksama kontinuiranog postavljanja i isporuke za dostavljanje koda krajnjim korisnicima nakon same provjere koda.[63] Navedene prakse kontinuirana integracija i kontinuirano postavljanje ili isporuka idu slijedno te zajedno omogućuju

[61] Usp. Isto (2023-06-15)
[62] Usp. Isto (2023-06-15)
[63] Usp. Isto (2023-06-15)

automatsku provjeru koda i ručno ili automatsko postavljanje provjerenog koda do krajnjeg korisnika. Time olakšavaju kontrolu valjanosti koda postavljenog na repozitorij i olakšavaju proces postavljanja aplikacije krajnjem korisniku sa svrhom uštede resursa izbjegavanjem više količine grešaka na kodu koji je postavljen na repozitorij te krajnjem korisniku.[64] Kako je i ranije navedeno prakse kontinuirane metodologije u praktičnom smislu odvijaju se slijedno. Prvotno se izvodi kontinuirana integracija, a zatim postavljanje aplikacije do krajnjeg korisnika. Takav oblik izvedbe u terminologiji na engleskom naziva se *pipeline*, a sam direktan prijevod budući da je nezgrapan i glasio bi cjevovod, preveden je kao linearni slijed događaja.[65]

3.1. Linearni slijed događaja

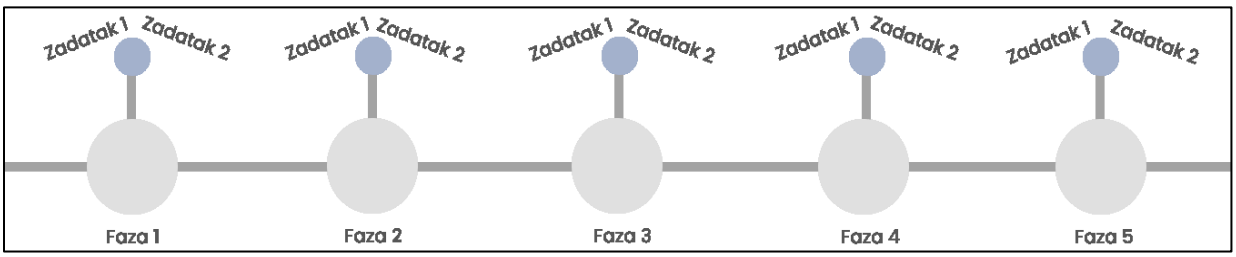
Semantika, znanost je koja se bavi određivanjem značenja riječi u pojedinoj rečenici.. Prema Hrvatskoj enciklopediji semantika se dijeli na dva temeljna područja istraživanja: odnos znaka prema svijetu i odnos znaka prema drugom znaku.[66] Odnosno semantika se bavi različitim značenjima koje određene riječi ili znakovi mogu imat u različitim kontekstima. Sama engleska riječ *pipeline* kao takva direktno prevedena s engleskog jezika bez ikakvog konteksta u hrvatskome jeziku označava cjevovod, pojam koji samoopisno predstavlja skup cijevi. No, kao što semantika naznačuje, određene riječi ovisno o kontekstu mogu imat potpuno drugačije značenje.[67] Stoga riječ *pipeline*, preciznije rečeno pojam, u kontekstu IT-a označava logički redoslijed instrukcija za izvođenje određenog zahtjeva. Navedena definicija *pipeline-a* u IT-u, kao takva, generalizirana je za sve kontekste IT-a i nedovoljno objašnjava aspekte vezane za kontinuiranu metodologiju, što je i očekivano budući da je ipak generalizirana definicija.[68] Stoga za preciznije definiranje pojma *pipeline,* u kontekstu kontinuirane metodologije, Gitlab je postavio definiciju *pipeline-a* kao pojam koji opisuje prijenos konstrukcije koda kroz različite faze sve do isporuke same aplikacije. Definicija kao takva slična je onome što sama kontinuirana metodologija predstavlja i prikladniji prijevod samog *pipeline-a* prema toj definiciji bio bi linearni slijed događaja.[69] Kako je bit

[64] Usp. Isto (2023-06-15)
[65] Usp. Isto (2023-06-15)
[66] Usp. Semantika. URL: https://enciklopedija.hr/natuknica.aspx?ID=55330 (2023-06-15)

[67] Usp. Isto (2023-06-15)
[68] Usp. Pipeline. URL: https://www.techopedia.com/definition/5312/pipeline (2023-06-15)

[69] Usp. A quick guide to GitLab CI/CD pipelines. URL: https://about.gitlab.com/blog/2019/07/12/guide-to-ci-cd-pipelines/ (2023-06-15)

linearnog slijeda događaja kroz različite faze provesti kod sve do objave on se sastoji od dva temeljna dijela *stage* i *job*.



Slika 11. *Grafički prikaz dijelova linearnog slijeda događaja*

Stage prevedeno faza dio je linearnog slijeda događaja koja definira kad se određeni *job,* odnosno zadatak provodi. Svaka faza može imat više definiranih zadataka čija je svrha definirati što se izvodi s kodom u određenoj fazi. Kako se i vidi na slici 11 jedan *stage*, tj. jedna faza u slijedu događaja definira jedan ili više *Job-ova*, odnosno zadataka koji određuju što se s kodom u određenoj fazi događa.[70] Zadatak kao sam pojam odnosi se na napisane skripte koje nam definiraju što učiniti s kodom u kojoj fazi. Ovisno o alatu kojeg koristimo za definiranje skripti poslova u linearnom slijedu događaja sama sintaksa skripti je drugačija.[71] Postoji znatan broj alata koji nude implementaciju kontinuirane metodologije a neki od njih su: Jenkins, TeamCity, CircleCI, Bamboo i Gitlab CI/CD. Budući da se svaki od navedenih servisa i alata koristi u različite svrhe i pisali su ih različiti ljudi, same skripte i mogućnosti implementacije kontinuirane metodologije variraju, ali ono što je uvijek zajedničko svakom, je da prilikom izgradnje principa kontinuirane metodologije, omogućuju implementaciju temeljnih ideja i principa uporabe kontinuirane integracije i kontinuiranog postavljanja.[72] Budući da je sam projekt smješten na Gitlab-ov repozitorij, za daljnje primjere koristi se Gitlab-ova konfiguracija sintakse za linearni slijed događaja. Budući da, kako je i ranije navedeno, sami poslovi unutar faze u linearnom slijedu događaja su skripte koje definiraju što će se provesti s kodom u određenoj fazi, kako bi se one i provele potrebno je konfigurirati program *Gitlab Runner* i *Runner Executor*.[73]

[70] Usp. CI/CD Pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/ (2023-06-15)
[71] Usp. Isto (2023-06-15)
[72] Usp. Best 14 CI/CD Tools You Must Know. URL: https://katalon.com/resources-center/blog/ci-cd-tools
[73] Gitlab Runner. URL: https://docs.gitlab.com/runner/ (2023-06-15)

3.2. Gitlab Runner i Runner Executor

Proces stvaranja i pokretanja linearnog slijeda je složen i zahtijeva potpuno razumijevanje svih segmenata tog slijeda. Jedan od ključnih elemenata procesa linearnog slijeda u Gitlabu je Gitlab Runner. Uz faze i poslove, koji su definirani ranije, neizostavan dio procesa linearnog slijeda Gitlab-a je *Gitlab Runner*. Prema Gitlabovoj dokumentaciji, Gitlab Runner je aplikacija koja izvršava zadatke u sklopu linearnog slijeda događaja.[74] Svaki Runner ima izvoditelja (eng. *Runner executor*) čija je svrha pokretanje zadatka. *Executor* može se definirati kao okruženje koje koristi različite tehnologije za pokretanje skripti koje definiraju pojedini zadatak. Budući da se različite tehnologije koriste za pokretanje zadataka, *executori* pružaju različite mogućnosti i kontekstualnu upotrebu za izvršavanje tih zadataka. Gitlab CI/CD omogućuje konfiguraciju različitih *executora*, ovisno o specifičnim potrebama tehnologije koja se koristi.[75] Opcije executora u Gitlabu uključuju: SSH(eng. Secure Socket Shell), Shell, VirtualBox, Parallels, Docker Machine, Docker i Kubernetes. SSH executor dopušta slanje naredbi preko SSH-a protokola do računala na kojem je runner instaliran.[76] Budući da SSH protokol daje određenu razinu sigurnosti prilikom slanja naredbi poželjno je koristiti SSH kada se sam runner ne nalazi na istom računalu gdje su i skripte za definiranje zadataka. Sličan executor SSH je Shell s razlikom u tome što Shell direktno pokreće naredbe na računalu gdje je i sam runner instaliran te je poželjan za korištenje kada želimo osigurati da kod koji se provodi kroz linearni slijed događaja je napisan za aplikaciju koja je namijenjena za specifičan operativni sustav.[77] VirtualBox i Parallels omogućuju pokretanje zadataka linearnog slijeda događaja u virtualnom okruženju. Navedeni programi simuliraju očvrsje računala, stvarajući potpuno virtualni računalni sustav na kojem se u kontekstu Gitlab CI/CD-a mogu pokrenuti zadaci. Ta funkcionalnost je korisna prilikom razvoja aplikacija koje se trebaju testirati na različitim operativnim sustavima jer proces virtualizacije omogućuje jednostavno stvaranje simulacija različitih operativnih sustava radi provođenja testova.[78] Nadalje Docker, u kontekstu

[74] Usp. Isto (2023-06-15)
[75] Usp. Isto (2023-06-20)
[76] Usp. Executors. URL: https://docs.gitlab.com/runner/executors/ (2023-06-20)
[77] Usp. Despa, Valentina. A Brief Guide to GitLab CI Runners and Executors, 2021. URL: https://medium.com/devops-with-valentine/a-brief-guide-to-gitlab-ci-runners-and-executors-a81b9b8bf24e (2023-20-06)
[78] Usp. Brush, Kate. What is Virtualization. URL: https://www.techtarget.com/searchitoperations/definition/virtualization (2023-20-06)

executora, odnosi se na program koji omogućava upotrebu i izradu Linux kontejnera.[79] Kontejner u vidu informacijske tehnologije odnosi se na set jednog ili više procesa koji su potpuno izolirani od ostatka sustava i sve datoteke koje su potrebne za pokretanje jednog kontejnera dostupne su s udaljenog izvora. U praktičnom smislu kontejneri nam omogućuju pohranjivanje svih potrebnih knjižnica koda, ovisnosti i drugih zahtjeva potrebnih za izradu aplikacije, koji će neovisno o sustavu na kojem jedan programer radi, lokalno omogućiti standardizirani rad na aplikaciji bez potrebe samostalnog preuzimanja svake pojedinačne ovisnosti na lokalni sustav za rad.[80] Docker odvaja procese u kontejneru od onih u sustavu te pritom koristi Linux kernel. Linux kernel glavna je komponenta Linux operacijskog sustava koji služi za komunikaciju između očvrsja i programa na sustavu, odnosno upravlja alokacijom memorije, procesima, driverima i sigurnošću sustava.[81] Stoga u kontekstu linearnog slijeda događaja Docker nam omogućuje pokretanje zadataka kroz određenu okolinu čije specifikacije možemo preuzeti iz Docker kontejnera. Docker Machine sličan je kao i standardni Docker. Alat je kreiran od Docker zajednice i omogućuje korisnicima postavljanje Dockera na vlastitom računalu, serveru ili vlastitom centru podataka, a služi za kreiranje servera i instaliranje Dockera na njima.[82] Nadalje Kubernetes je platforma za upravljanje kontejnerima. Automatizira procese upravljanja, postavljanja i skaliranja kontejneriziranih aplikacija. Za razumijevanje upotrebe Kubernetesa prilikom izvođenja poslova važno je razumjeti i samu arhitekturu i način na koji funkcionira Kubernetes. Kubernetesov glavni koncept je *Kubernetes cluster*. Cluster, prevedeno s engleskog označava grupu, a u kontekstu *Kuberntes cluster-a* odnosi se na grupiranje mašina koje se u Kubernetesu nazivaju *nodes* i služe za upravljanje kontejneriziranim aplikacijama.[83] Na navedenim mašinama unutar *Kubernetes cluster-a* mogu se pokrenuti zadaci iz linearnog slijeda događaja. Naravno sama arhitektura Kubernetes-a je znatno kompleksnija i njegova upotreba je široka, a za potrebe executora dovoljno je razumjeti

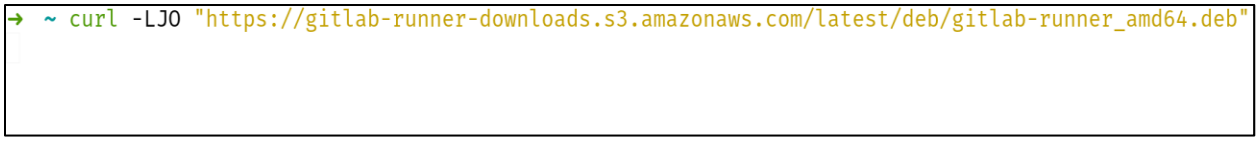
[79] Usp. Despa, Valentina. A Brief Guide to GitLab CI Runners and Executors, 2021. URL: https://medium.com/devops-with-valentine/a-brief-guide-to-gitlab-ci-runners-and-executors-a81b9b8bf24e (2023-20-06)
[80] Usp. What's a Linux container?. URL: https://www.redhat.com/en/topics/containers/whats-a-linux-container (2023-06-20)
[81] Usp. What is the Linux kernel? URL: https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel (2023-06-20)
[82] Usp. Despa, Valentina. A Brief Guide to GitLab CI Runners and Executors, 2021. URL: https://medium.com/devops-with-valentine/a-brief-guide-to-gitlab-ci-runners-and-executors-a81b9b8bf24e (2023-06-20)
[83] Usp. Kubernetes Components. URL: https://kubernetes.io/docs/concepts/overview/components/ (2023-06-20)

općenitu arhitekturu.[84] Navedene executore korisnik odabire prilikom samostalne konfiguracije Gitlab runner-a. Gitlab korisniku omogućuje instalaciju i potpunu konfiguraciju na vlastitom računalu ili serveru gdje korisnik koristi vlastite resurse prilikom pokretanja Gitlab runner-a.

3.2.1. Instalacija i konfiguracija Gitlab Runner-a

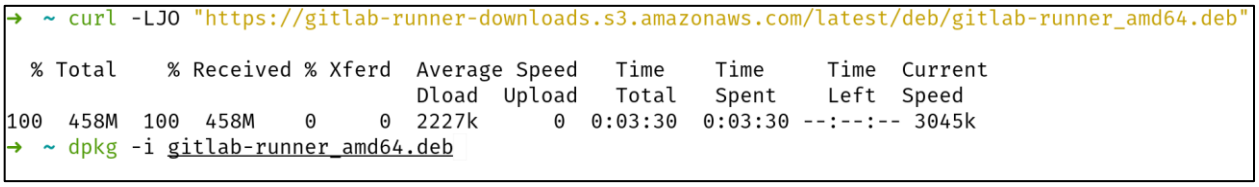
GitLab platforma nudi dva različita pristupa korištenja runner-a. Prvi pristup obuhvaća unaprijed konfigurirane runner-e smještene na GitLab-ovom SaaS (eng. Software as a Service) okruženju. Runner-i na SaaS-u potpuno su integrirani s Gitlab platformom te su prema početnim postavkama omogućeni za svaki projekt korisnika.[85] Naravno ukoliko se korisnik odluči koristiti Gitlab-ovim SaaS Runner-ima dužan je platiti naknadu koja se računa prema tome koliko pojedinačni zadatak treba vremena da se izvede pomnoženo s Gitlab-ovim konceptom utjecaja na trošak (eng. Cost factor), koji uključuje različite čimbenike poput broja projekta koji koriste runner, veličini računala na kojem je SaaS runner postavljen i svi ostali čimbenici koji su naznačeni na Gitlab-ovoj stranici. Plaćanjem usluge SaaS runner-a korisnik se ne mora brinuti o konfiguraciji niti o postavljanju executor-a i dobija potpuno siguran runner koji može koristiti na svim svojim projektima.[86] Ukoliko korisnik ne želi plaćati Gitlab-u SaaS runner-e ili jednostavno želi veću kontrolu nad runner-om koji mu pokreće zadatke u linearnom slijedu događaja, ima mogućnost konfiguracije vlastitog runner-a. Kako je naznačeno ranije, Gitlab korisniku omogućuje instalaciju i potpunu konfiguraciju runner-a na vlastitom računalu ili serveru. Proces instalacije započinje preuzimanjem datoteka za instalaciju s Gitlab-a. Ovisno o operacijskom sustavu računala potrebno je preuzeti datoteku za instalaciju koja odgovara sustavu.[87] Budući da se sve aktivnosti prikazane u diplomskom radu odvijaju na Linux operacijskom sustavu, sam primjer instalacije i daljnje konfiguracije prikazan je na Linux-u, a za ostale operacijske sustave potrebno je pratiti uputstva u Gitlab-ovoj dokumentaciji.[88]

[84] Usp. Kubernetes Components. URL: https://kubernetes.io/docs/concepts/overview/components/ (2023-06-20)
[85] Usp. Runner SaaS. URL: https://docs.gitlab.com/ee/ci/runners/index.html (2023-06-20)
[86] Usp. Isto (2023-06-20)
[87] Usp. Install Gitlab Runner. URL: https://docs.gitlab.com/runner/install/index.html (2023-06-20)
[88] Usp. Install GitLab Runner manually on GNU/Linux. URL: https://docs.gitlab.com/runner/install/linux-manually.html (2023-06-20)



Slika 11. *Prikaz naredbe za preuzimanje Gitlab Runner-a u naredbenom sučelju*

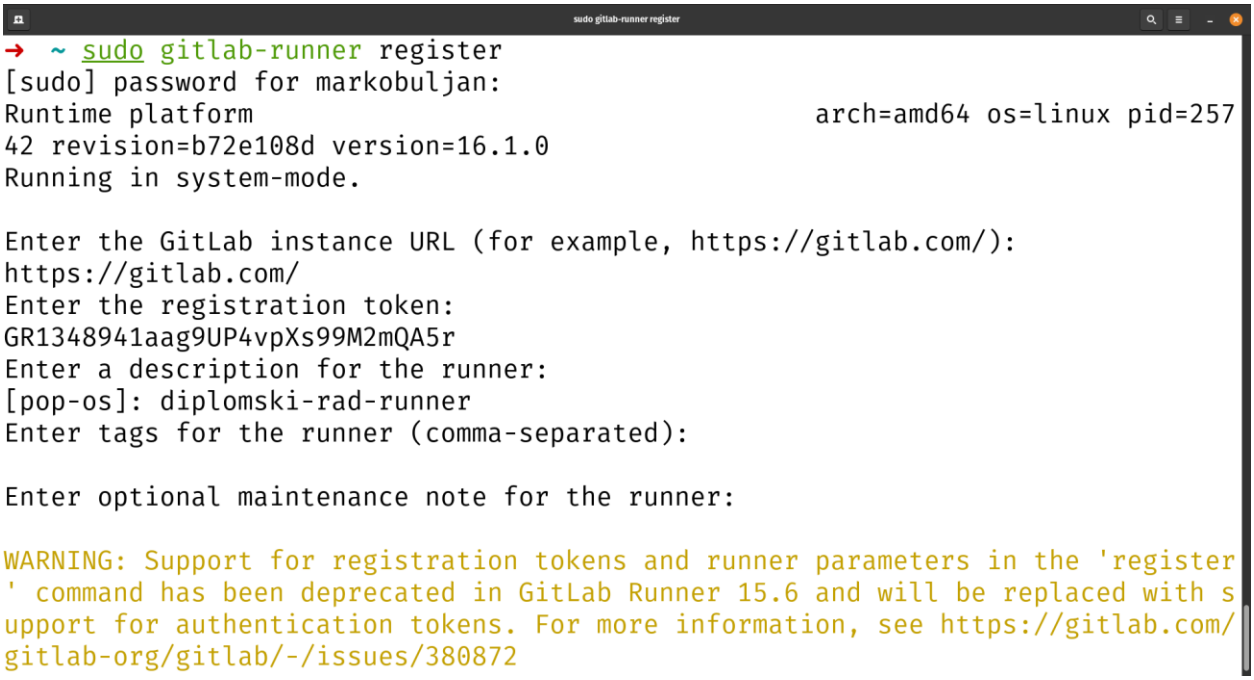
Naredba na slici 11 u naredbenom sučelju preuzima Gitlab Runner na Linux sustav. Ono što je potrebno prije puštanja naredbe je varijablu ${arch} zamijeniti s nekim od podržanih arhitektura koje se nalaze na dokumentaciji gdje korisnici Windows i Mac operativnog sustava mogu direktno preuzeti Runner s pojedinačnom arhitekturom koja odgovara njihovom sustavu. [89]



Slika 12. *Prikaz naredbe za instalaciju Gitlab Runner-a u naredbenom sučelju*

Nakon preuzimanja Runner-a s amd64 arhitekturom instalira se navedenom naredbom s istom arhitekturom te nakon instalacije slijedi konfiguracija. Registracija Runner-a je proces povezivanja lokalno instaliranog Runner-a s instancom projekta na kojoj želimo koristiti instalirani Runner. Prema temeljnim postavkama lokalno instalirani Runner se prvotno registrira samo za jedan projekt, ali kroz sučelje Gitlab-a može se konfigurirati i kao dijeljeni runner što će se i prikazati u kasnijim primjerima. [90]

[89] Usp. Isto (2023-06-20)
[90] Usp. Isto (2023-06-20)



Slika 13. *Prikaz provedbe registriranja Gitlab Runner-a kroz naredbeno sučelje*

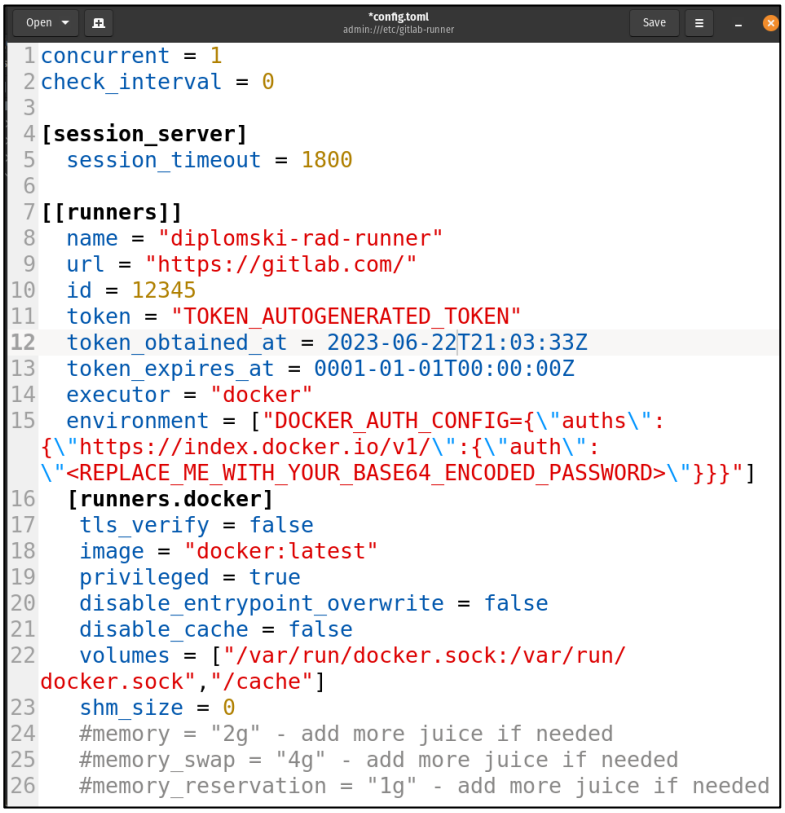
Naredba za registriranje Gitlab Runner-a, kao što se može vidjeti na slici 13 je *gitlab-runner register* te se nakon nje otvara postupak registriranja u koje se dodaju instanca Gitlab-a, zaseban token pojedinačnog projekta koji se jednostavno može generirati kroz sučelje Gitlab-a ulaskom na postavke profila, izbornik naziva CI/CD i podizbornik Runner, nudi generaciju zasebnog tokena s kojim se lokalni Runner povezuje s projektom.[91] Navedene dvije opcije konfiguracije najbitnije su za povezivanje lokalno instaliranog Gitlab Runner-a s projektom na Gitlab-u jer se njime zapravo i sama poveznica stvara. Osim spajanja važno je i izabrati executor-a te ovisno o potrebama projekta može se izabrati jedan od već ranije navedenih executor-a te je u primjeru odabran docker.[92] Ono što je potrebno dodatno napraviti nakon same registracije Runner-a je konfiguracija automatski generirane datoteke *config.toml* budući da bez dodatne konfiguracije testovi koji se provode nad kodom se neće pokrenuti kroz Docker *executor*.[93] Prije konfiguracije potrebno je izraditi korisnički račun na Docker-u te korisničko ime i lozinku u kombinaciji enkodirati formatom base64 za enkodiranje i dekodiranje znakova.

[91] Usp. Isto (2023-06-20)
[92] Usp. Isto (2023-06-20)
[93] Usp. Advanced configuration. URL: https://docs.gitlab.com/runner/configuration/advanced-configuration.html (2023-06-20)

```
➜  ~ echo -n "username:password" | base64
```

Slika 14. *Prikaz enkodiranja lozinke i korisničkog imena kroz naredbeno sučelje*

Na slici 14 prikazana je naredba za enkodiranje lozinke i korisničkog imena kroz naredbeno sučelje gdje se jednostavno upišu korisničko ime i lozinka docker-a te se dobije base64 enkodirani set znakova.[94] Navedeni set znakova potrebno je ukomponirati u config.toml datoteku.

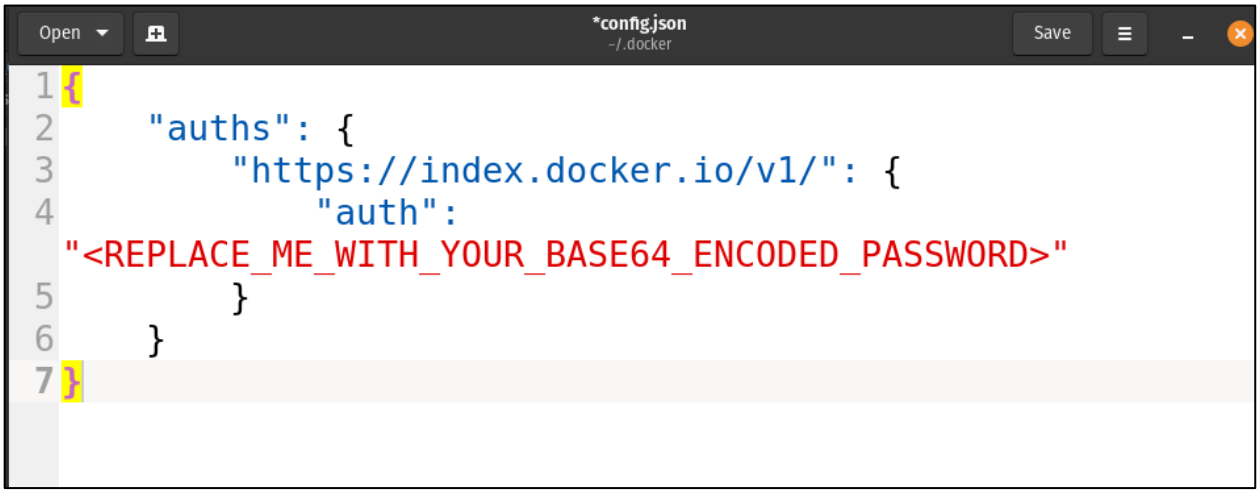


Slika 15. *Prikaz konfiguracije Gitlab Runner-a*

Set znakova generiran naredbom, prikazanom na slici 14, potrebno je zamijeniti u *environment* varijabli u config.toml datoteci. U praktičnom smislu sama zamjena vrijednosti *environment* varijable prikazana je na slici 15 te vrijednost token varijable automatski je ispisana, no radi zaštite

[94] Usp. Yesmin, Fahmida. Bash base64 encode and decode. URL: https://linuxhint.com/bash_base64_encode_decode/ (2023-06-25)

privatnosti i sigurnosti sam generirani set znakova nije prikazan na slici 15.[95] Unutar same datoteke moguće je konfigurirati i ostale mogućnosti, primjerice količina radne memorije koju runner koristi i ostale mogućnosti vezane za sklopovlje i resurse uređaja koje su runneru dostupni na korištenje, ali je nakon konfiguracije potrebno resetirati runner naredbom gitlab runner restart.[96] Nakon dodavanja enkodiranog seta znakova docker korisničkog imena i lozinke te ukoliko je potrebno dodatnog konfiguriranja runner-a važno je enkodirani set znakova dodati i u konfiguraciju docker-a kako bi executor prilikom pokretanja automatski autentificirao vaš docker korisnički račun. Config.json datoteka docker-a nalazi se u .docker mapi.[97]

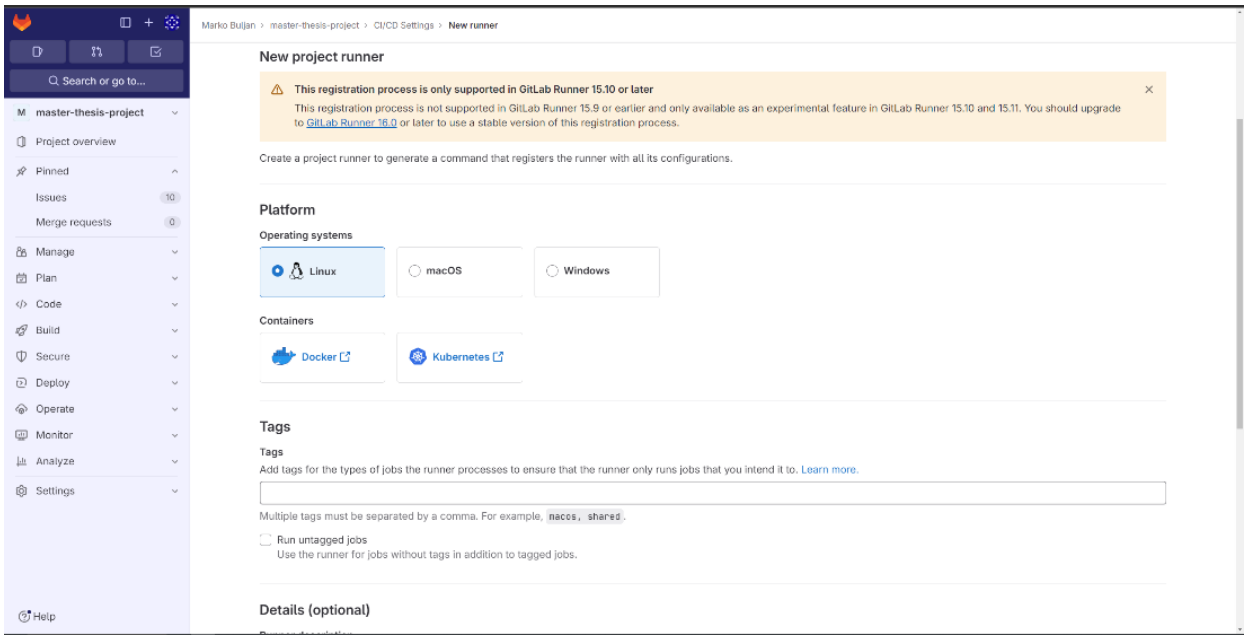


Slika 16. *Prikaz konfiguracije docker-a*

Prikazano na slici 16 je i sama config.json datoteka u koju je potrebno na odgovarajuće mjesto dodati enkodirani set znakova korisničkog imena i lozinke docker-a, kako bi sve potrebne kontejnere mogli koristiti s Docker-a.[98] Iako su navedeni koraci dovoljni za postavljanje runner-a s Docker executor-om, važno je napomenuti da se u dinamičnom području informacijske tehnologije često uvode promjene radi poboljšanja sigurnosti, strukture i korisničkog iskustva. U skladu s tim, proces konfiguracije runner-a prema opisu označenom žutom bojom na slici 13, koji je važio do Gitlab verzije 15.6, sada se ne preporučuje za korištenje. Razloga za postavljanje novog načina instalacije i registracije lokalnog runner-a prema Gitlab-ovoj dokumentaciji ima nekoliko,

[95] Usp. Advanced configuration. URL: https://docs.gitlab.com/runner/configuration/advanced-configuration.html (2023-06-25)
[96] Usp. Isto (2023-06-25)
[97] Usp. Lushpenko, Maksym. Creating Docker config.json for external systems. URL: https://lumaks.medium.com/creating-docker-config-json-for-use-in-external-systems-a5d14aa81c41 (2023-06-25)
[98] Usp. Isto (2023-06-25)

a glavni razlog je omogućavanje korisnicima korištenje jednog tokena za registraciju više runner-a.[99] Prijašnji, već prikazani oblik instalacije i registracije runner-a jednostavno je povezivao projekt na repozitoriju s lokalnim runner-om preko zasebnog identifikacijskog skupa znakova projekta kojeg nazivamo token projekta. Novi proces registracije i inicijacije lokalnog runner-a, za razliku od prijašnjeg koji koristi zasebni token projekta za povezivanje lokalnog runner-a i projekta, koristi token korisničkog računa kojeg prilikom registracije sprema u konfiguraciju i provjerava njegovu valjanost.[100] Time nova metoda u samom Gitlab-ovom sustavu smanjuje broj token-a potrebnih za upravljanje, a korisnicima omogućuje fleksibilnije upravljanje runner-ima korištenjem jednog token-a za registraciju više runner-a i jednostavniju izradu samog Gitlab-ovog runner-a budući da njegova inicijacija započinje kroz grafičko sučelje.[101]

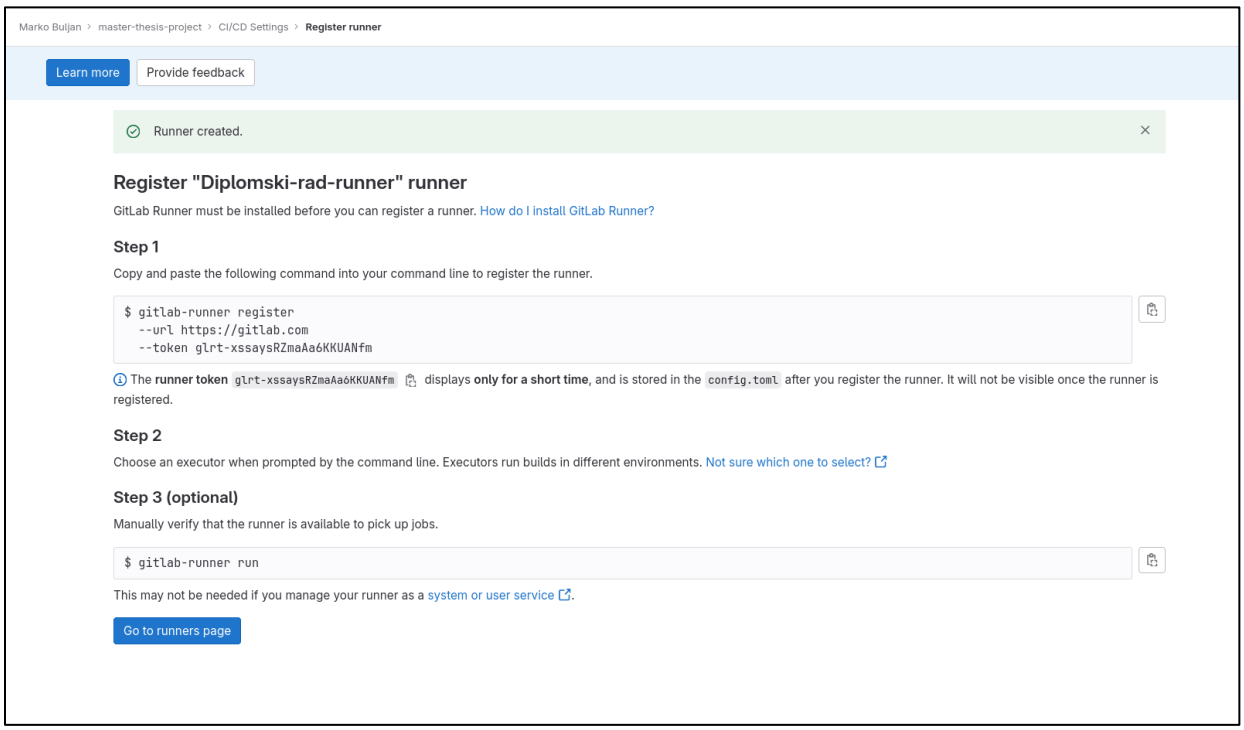


Slika 17. *Prikaz inicijacije runner-a kroz Gitlab sučelje*

Prikazano slikom 17 vidi se početak procesa inicijacije Gitlab-ovog runner-a kroz grafičko sučelje Gitlab-a. Iniciranjem runner-a kroz sučelje korisnik odabire na kojem operacijskom sustavu će biti

[99] Usp. Migrating to the new runner registration workflow. URL: https://docs.gitlab.com/ee/ci/runners/new_creation_workflow.html (2023-07-03)
[100] Usp. Isto (2023-07-03)
[101] Usp. Isto (2023-07-03)

instaliran lokalni runner, hoće li kupiti označene ili neoznačene zadatke te dodjeljuje mu kratki opis ukoliko želi.[102]



Slika 18. *Prikaz koraka nakon inicijacije runner-a*

Nakon što korisnik uspješno inicira runner, nastavlja proces registracije kao i ranije prikazanim načinom, a to je kroz naredbeno sučelje gdje jednostavno prati upute s Gitlab-ove stranice. Upisuje naredbu za registraciju prikazanu na slici 18 te budući da su sve ostale opcije konfiguracije odabrane prilikom iniciranja runner-a, korisnik jedino mora izabrati executor-a te provjeriti radi li sam runner.[103] Za dodatno konfiguriranje, primjerice postavljanje docker sustava okoline za izvođenje poslova potrebno je kao i ranijim prikazom konfigurirati opcije u konfiguracijskog datoteci koja za korisnika ostaje nepromijenjena. Kako će se stariji proces inicijacije potpuno ukloniti u budućnosti, preciznije od Gitlab-ove verzije 17.0, važno je za naglasiti da je na demonstrativnom projektu korišten runner konfiguriran novijim procesom konfiguracije radi ažurnosti samog rada.[104] Nakon konfiguracije i iniciranja projekta za korištenje runner-a potrebno

[102] Usp. Isto (2023-07-03)
[103] Usp. Isto (2023-07-03)
[104] Usp. Isto (2023-07-03)

je svaki put pokrenuti sam runner lokalno naredbom kroz naredbeno sučelje *gitlab-runner run* te naravno napisati skriptu sa zadacima koju bi sam runner i iščitavao.[105]

3.3. Definiranje Gitlab zadataka i varijabli u linearnom slijedu događaja

S teorijskog pristupa, zadatak, kako je ranije definirano, dio je faze linearnog slijeda te opisuje što se u određenoj fazi s kodom odvija.[106] Varijable, kao i u programskim jezicima služe za spremanje pojedinih podataka koje želimo ponovno iskoristiti. Spremljene varijable koriste se za upravljanje pokretanja zadatka u linearnom slijedu te za sprječavanje ponavljanja iste vrijednosti za sve grane.[107] Naravno u samom Gitlab-u postoji niz unaprijed definiranih varijabli, ali mogu se i posebno definirati. U konfiguracijskoj datoteci varijable se definiraju unutar bloka koji se otvara ključnom riječi *variables*, a same varijable definiraju se nazivanjem varijable unutar bloka velikim slovnim znakovima te upisivanjem podatka kojeg varijabla sprema iza dvotočke. Mogu se definirati unutar zadatka, čije se definiranje detaljnije obrađuje u ostatku podnaslova, na početku konfiguracije. Ukoliko su varijable definirane na početku konfiguracije svaki zadatak ih nasljeđuje prema temeljnim postavkama.[108]

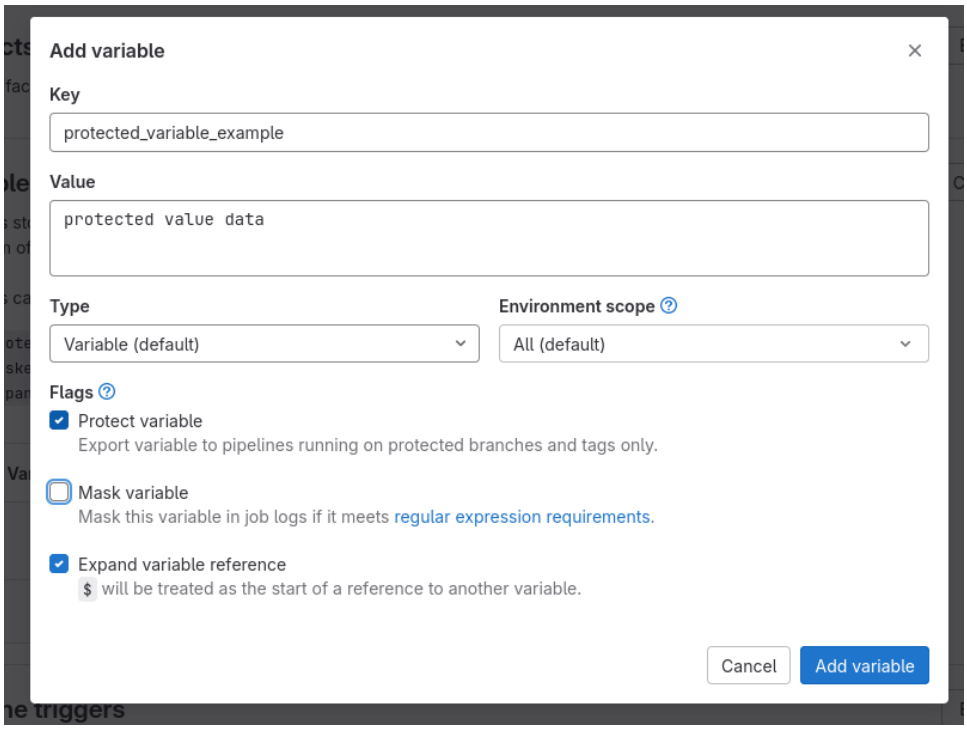
[105] Usp. Isto (2023-07-03)
[106] Usp. Jobs. URL: https://docs.gitlab.com/ee/ci/jobs/ (2023-07-14)
[107] Usp. GitLab CI/CD variables. URL: https://docs.gitlab.com/ee/ci/variables/ (2023-07-14)
[108] Usp. Isto (2023-07-14)

```yaml
variables:
  # Use TLS https://docs.gitlab.com/ee/ci/docker/using_docker_build.html#tls-enabled
  DOCKER_HOST: tcp://docker:2376
  DOCKER_TLS_CERTDIR: "/certs"
  IMAGE_TAG: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG

app:test:
  stage: test
  script:
    - echo "Testing code ..."
    - ./mvnw verify

app:docker:
  only:
    - main
    - /^deploy\/.*$/
  stage: docker
  image: docker:dind
  services:
    - name: docker:dind
      alias: docker
  script:
    - docker --version
    - docker login -u $CI_REGISTRY_USER -p $ACCESS_TOKEN $CI_REGISTRY
    - docker build -t $IMAGE_TAG .
    - docker push $IMAGE_TAG
```

Slika 19. *Prikaz definiranja varijabli i korištenja u zadatku*

Globalne varijable mogu se izostaviti iz pojedinačnog zadatka korištenjem njihovog naziva i dodjeljivanjem vrijednosti vitičastih zagrada, a pozivaju se stavljanjem znaka dolara zajedno s nazivom varijable, kao što se vidi i na slici 19, gdje se u varijable spremaju podaci vezani za docker image i spajanje na docker image.[109] Budući da pojedine varijable mogu sadržavati sigurnosno osjetljive podatke, poput lozinki, token-a i korisničkih imena te bilo kojeg drugog podatka koji ne bi trebao biti dostupan svima, a kako je konfiguracijska datoteka dostupna za pregled u repozitoriju, Gitlab osim definiranjem varijabli kroz konfiguracijsku datoteku nudi korisnicima mogućnost definiranja varijabli i kroz sučelje Gitlab-a. One se mogu definirati za pojedinačni projekt, za sve projekte u pojedinačnoj grupi te za sve projekte pojedinačne Gitlab instance. [110]

[109] Usp. Isto (2023-07-14)
[110] Usp. Isto (2023-07-14)

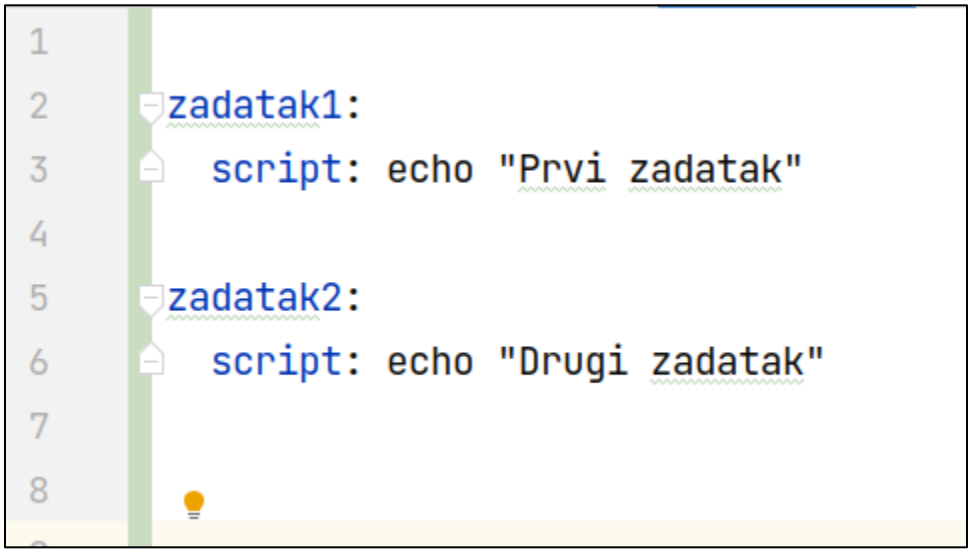


Slika 20. *Prikaz definiranja zaštićene varijable kroz Gitlab sučelje*

Varijable se praktično kroz sučelje definiraju odlaskom u postavke ili samog projekta, grupe te instance unutar izbornika, CI/CD, gdje Gitlab nudi izradu varijabli. Na primjeru 20 Prikazan je primjer izrade varijable kroz sučelje. Sam naziv mora sadržavati slovne ili brojne znakove i donju crtu.[111] Dodatne opcije konfiguracije omogućuje obilježavanje varijable kao zaštićene varijable gdje se i sama varijabla pokreće samo na zaštićenoj grani te se može maskirati da se ne prikazuje u logovima.[112] Nadalje, kao sam pojam, zadatak, odnosi se na napisane skripte koje nam definiraju što učiniti s kodom u određenoj fazi linearnog slijeda događaja. Time su zadaci zapravo i njegova temeljna komponenta.[113] Ujedno su i najmanja komponenta, budući da se uz pomoć zadataka, detaljno definira svaki proces koji se izvršava na kodu aplikacije, što primjerice mogu biti: provjera uspijeva li napisani kod izgraditi aplikaciju, prolazi li kod definirani testove, provjera pokrivaju li testovi potpuno cijelu aplikaciju i brojni drugi procesi, koji se definiraju prema potrebama projekta nad aplikacijom.[114] U praktičnom smislu Gitlab zadaci definiraju se u datoteci naziva *.gitlab-ci* YAML(eng. Yet Another Markup Language ili Ain't Markup Language) formata. YAML je jezik

[111] Usp. Isto (2023-07-14)
[112] Usp. Isto (2023-07-14)
[113] Usp. Isto (2023-07-14)
[114] Usp. Isto (2023-07-20)

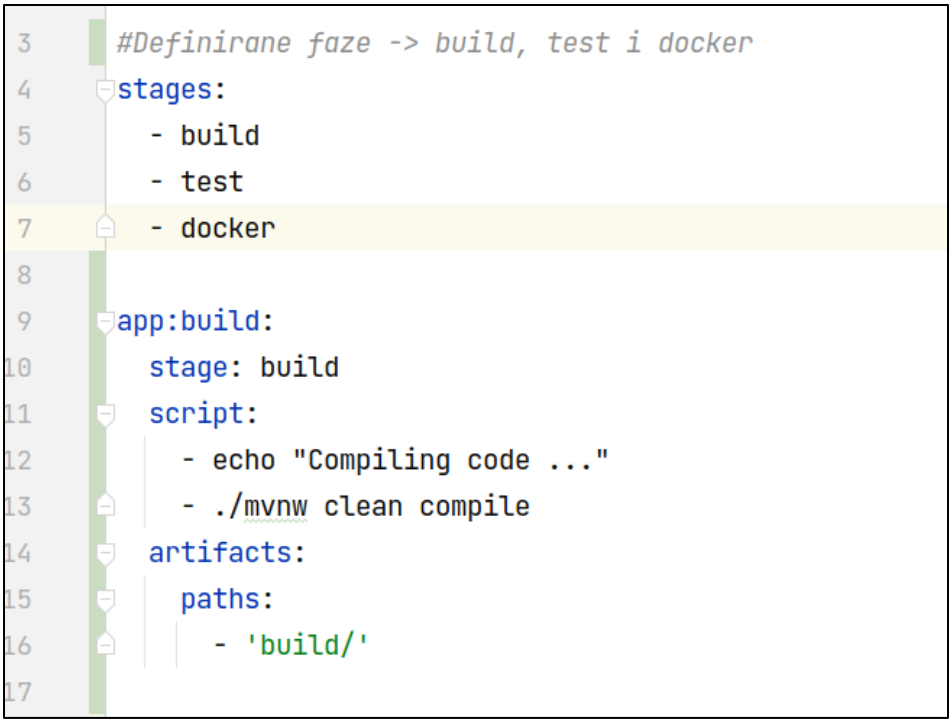
za serijalizaciju podataka.[115] Jedna od njegovih glavnih vrlina je činjenica da je svojom sintaksom lako čitljiv i pristupačan za korisnike. Budući da je lako čitljiv i pristupačan često se koristi uz programske jezike za dodatnu konfiguraciju što je slučaj i sa samom konfiguracijom zadataka i faza linearnog slijeda događaja u Gitlab-ovom sustavu.[116] Zadaci se u Gitlab-u uvijek definiraju sa svojevoljno odabranim nazivom i Gitlab-ovim elementom *script*.



Slika 21. *Prikaz jednostavne YAML sintakse Gitlab zadatka*

Prikazano slikom 21 vide se definirana dva zadatka jednostavnih naziva zadatak1 i zadatak2. Prilikom nazivanja svakog zadatka važno je znati da naziv može biti do 255 znakova i sam naziv ne može biti jednak kao neki od Gitlab-ovih elemenata, od kojih je jedan i već navedeni *script*. Element *script* koristi se za specificiranje naredbe ili akcije koja će biti izvedena.[117] U jednostavnom prikazanom primjeru na slici 21 to je jednostavno ispisivanje napisanih znakova unutar navodnika. Kako se sami zadaci provode unutar faza u linearnom slijedu događaja, potrebno je za svaki zadatak definirati faze.[118]

[115] Usp. What is YAML?. URL: https://www.redhat.com/en/topics/automation/what-is-yaml (2023-07-23)
[116] Usp. Isto (2023-07-23)
[117] Usp. Isto (2023-07-23)
[118] Usp. Isto (2023-07-23)



Slika 22. *Prikaz fazi i zadatka s fazom u konfiguraciji linearnog slijeda događaja*

Same faze, u Gitlab-ovoj konfiguraciji linearnog slijeda događaja definiraju se s elementom naziva *stages*. Definiraju se u pravilu na početku same konfiguracije kako bi se kasnije mogle koristiti u samim zadacima.[119] Kao što se vidi na primjeru na slici 22 definirane su faze *build*, *test* i *docker* te svaka od navedenih faza može imati nekoliko zadataka. Na slici 22 prikazan je zadatak naziva *app:build* i ključnom rječju *stage* i dodavanje vrijednosti *build* navedenoj ključnoj riječi zadatak *app:build* dodan je u fazu *build*. Važno je i naglasiti da osim zadatka *app:build* u samoj fazi može biti više zadataka. Ono što je direktna svrha zadatka definirana elementom *script* u ovom slučaju je izvođenje naredbe za kompajliranje koda kroz Maven sustav.[120] Maven je sustav za upravljanje projektima i Java programskom jeziku te nam služi za izgradnju aplikacije, upravljanje ovisnostima te organizaciju dokumentacije. Nakon što se naredba *clean compile* izvrši ili ne izvrši uspješno, metapodaci o zadatku spremaju se u mapu naziva *build*, što je definirano Gitlab CI/CD elementima *artifacts,* koji nam služi za definiranje spremanja, artefakata, odnosno metapodataka o samom zadatku koji se izvodi te *elementom paths* kojim naznačujemo gdje ćemo spremiti

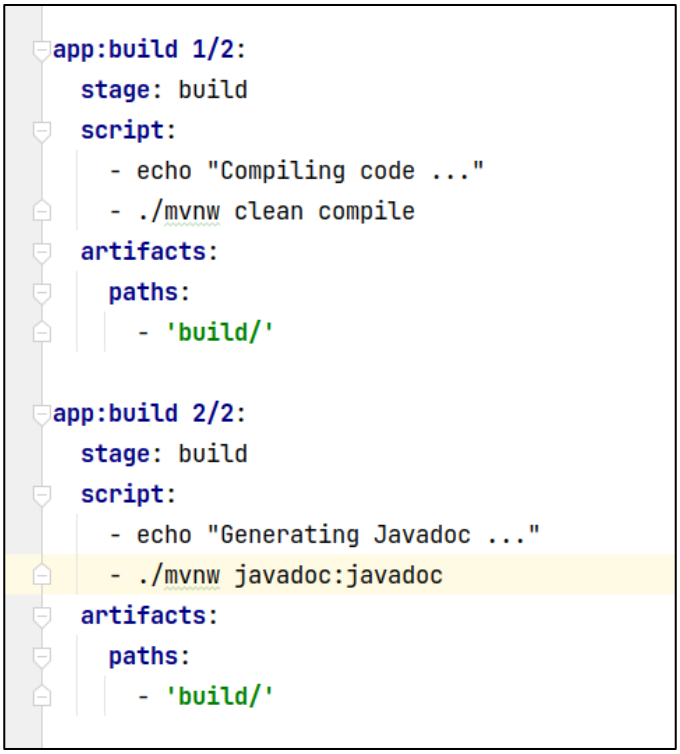
[119] Usp. Isto (2023-07-23)
[120] Usp. Isto (2023-07-23)

određeni sadržaj.[121] Artefakti samog zadatka spremaju se u repozitorij artefakata koji je neizostavan dio procesa linearnog slijeda događaja. Osigurava dosljednost i praćenje artefakata kroz različite faze i okoline u procesu, omogućujući jasan uvid u verziju koda koja je trenutno implementirana i gdje se nalazi u bilo kojem trenutku. Što nam pruža transparentnost prilikom prikaza podataka i pojednostavljuje proces otklanjanja pogrešaka.[122] Ujedno najvažnija prednost repozitorija artefakata je spremanje podataka o ovisnostima lokalno čime se potpuno uklanja potreba preuzimanja ovisnosti potrebnih za pokretanje aplikacije, što ubrzava sam linearni slijed događaja. Na samom primjeru slike 20. prilikom kompajliranja koda prvi put u izvođenju zadatka svi podaci o ovisnostima potrebnim za samu aplikaciju spremaju se u repozitorij te se više ne preuzimaju iz vanjskih izvora, već su dostupne u repozitoriju.[123] Nadalje kontrola pristupa i dopuštenja osigurava da samo ovlašteni članovi tima mogu pristupiti osjetljivim komponentama. Enkripcija i digitalni potpisi dodaju dodatni sloj zaštite kako bi se spriječile neovlaštene izmjene ili manipulacije. Sve navedeno čini artefaktorij nužnim dijelom linearnog slijeda događaja. Iako je artefaktori ključan dio linearnog slijeda događaja, organizacija zadataka i definiranje pokretanja zadataka koncepti su koji poput artefaktorija imaju ulogu u brzini izvođenja.[124]

3.3.1. Opcije grupiranja i pokretanja Gitlab zadataka u linearnom slijedu događaja

Budući da pojedinačna faza može imati nekoliko zadataka, njihovo grupiranje važno je za preglednost, kako sam vizualni graf za očitavanje zadataka ne bi bio nepregledan.[125]

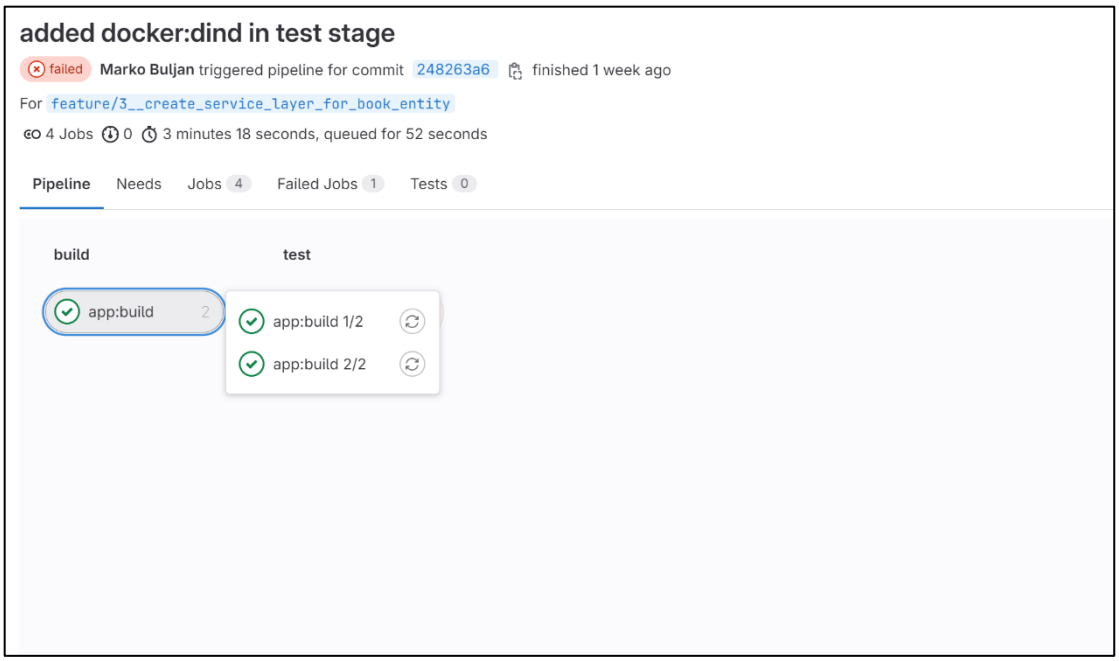
[121] Usp. What is Maven?. URL: https://maven.apache.org/what-is-maven.html (2023-07-23)
[122] Usp. Yakutovich, Anton. GitLab CI: Cache and Artifacts explained by example. URL: https://dev.to/drakulavich/gitlab-ci-cache-and-artifacts-explained-by-example-2opi (2023-07-23)
[123] Usp. Isto (2023-07-23)
[124] Usp. Isto (2023-07-23)
[125] Usp. Jobs: Group jobs in a pipeline. URL: https://docs.gitlab.com/ee/ci/jobs/#group-jobs-in-a-pipeline (2023-07-27)



Slika 23. *Prikaz grupiranja 2 zadatka u jednoj fazi*

Poslovi u jednoj fazi automatski se mogu grupirati definiranjem naziva poslova. Pravila koja se koriste pri imenovanju poslova koji se žele grupirati su korištenje brojeva i znakova kose crte, dvotočke ili samog razmaka između broja zadatka i ukupnog broja zadataka u jednoj fazi.[126] U primjeru na slici 23 korišten je znak kose crte za odvajanje poslova, gdje prvi zadatak u fazi naziva *build* pokreće, odnosno izgrađuje aplikaciju, a drugi dio generira java dokumentaciju.[127] Javadoc je alat koji dolazi s JDK (eng. *Java Development Kit*) i koristi se za generiranje dokumentacije iz izvornog Java koda. Dodavanje Javadoc-a u konfiguraciju može pomoći u održavanju ažurirane dokumentacije za Java projekt.[128]

[126] Usp. Isto (2023-07-27)
[127] Usp. Isto (2023-07-27)
[128] Usp. Javadoc Tool. URL: https://www.oracle.com/java/technologies/javase/javadoc-tool.html (2023-07-27)



Slika 24. *Prikaz vizualnog grafa linearnog slijeda događaja s grupiranim zadacima*

Navedeni prikazani zadaci u prikazu vizualnog grafa zbog korištenja pravila naziva postaju grupirani i lakši za čitanje što je i prikazano slikom 24 Iako na samom primjeru ima malo zadataka u linearnom slijedu događaja, gdje ima veći broj zadataka organizacija je nužna za preglednost.[129] Osim grupiranja, konfiguracijom možemo utjecati i na faktore vezane za pokretanje poslova u različitim fazama linearnog slijeda događaja. Element ili ključna riječ, koja se u konfiguracijskoj datoteci koristi za otvaranje bloka definiranja pravila, koji određuju uvjete za pokretanje pojedinačnog zadataka u linearnom slijedu događaja naziva se *rules.* Unutar samog bloka za određivanja uvjeta za pokretanje zadataka, koriste se uvjetni elementi za konfiguraciju *if i when.*[130]

[129] Usp. Jobs: Group jobs in a pipeline. URL: https://docs.gitlab.com/ee/ci/jobs/#group-jobs-in-a-pipeline (2023-07-27)
[130] Usp. Choose when to run jobs. URL: https://docs.gitlab.com/ee/ci/jobs/job_control.html (2023-07-27)

```yaml
app:docker:
  stage: docker
  image: docker:dind
  services:
    - name: docker:dind
      alias: docker
  script:
    - docker --version
    - docker login -u $CI_REGISTRY_USER -p $ACCESS_TOKEN $CI_REGISTRY
    - docker build -t $IMAGE_TAG .
    - docker push $IMAGE_TAG
  rules:
    - if: '$CI_COMMIT_REF_NAME == "main"'
      when: on_success
    - if: '$CI_COMMIT_REF_NAME =~ /^deploy\/.*$/'
      when: on_success
```

Slika 25. *Prikaz određivanja uvjeta pokretanja zadatka elementima if i when unutar rules elementa*

Kombiniranjem uvjetnih elemenata *if* i *when* postavljaju se uvjeti koje zadatak mora postići da bi bio pokrenut. U primjeru na slici 25 Unutar if elementa koristi se *$CI_COMMIT_REF_NAME* gitlab-ova varijabla u koju se dodaje naziv grane na kojoj se projekt izvodi, odnosno gradi, a u *when* elementu dodana vrijednost *on_success* koja naznačuje da će se navedeni zadatak pokrenuti samo ukoliko se linearni slijed događaja izvodi na git main grani ukoliko je aplikacija uspješno izgrađena.[131] *When* element osim navedenog još omogućuje definiranje ručnog pokretanja samog zadatka dodavanjem vrijednosti *manual*. Unutar samog if elementa mogu se koristiti i sve ostale Gitlab-ove unaprijed definirane varijable, ali i korisničke varijable za postavljanje uvjeta. Za definiranje uvjeta, osim *rules* elementa koriste se i elementi *only* i *except.* Važno je naglasiti da u 2023. godini prema Gitlab-ovoj dokumentaciji *only* i *except* nisu aktivno održavani te je preferirani način postavljanja uvjeta pokretanja pojedinog zadataka korištenjem elementa *rules*.[132] *Only* element koristi se za definiranje kada se zadatak pokreće, a *except* za definiranje slučaja kada se zadatak ne pokreće.

[131] Usp. Isto (2023-07-27)
[132] Usp. Isto (2023-07-27)

```
app:test:
  stage: test
  services:
    - docker:dind
  script:
    - echo "Testing code ..."
    - ./mvnw verify

app:docker:
  only:
    - main
    - /^deploy\/.*$/
  stage: docker
  image: docker:dind
  services:
    - name: docker:dind
      alias: docker
  script:
    - docker --version
    - docker login -u $CI_DEPLOY_USER -p $CI_DEPLOY_PASSWORD $CI_REGISTRY
    - docker build -t $IMAGE_TAG .
    - docker push $IMAGE_TAG
```

Slika 26. *Prikaz određivanja uvjeta pokretanja zadatka elementom only*

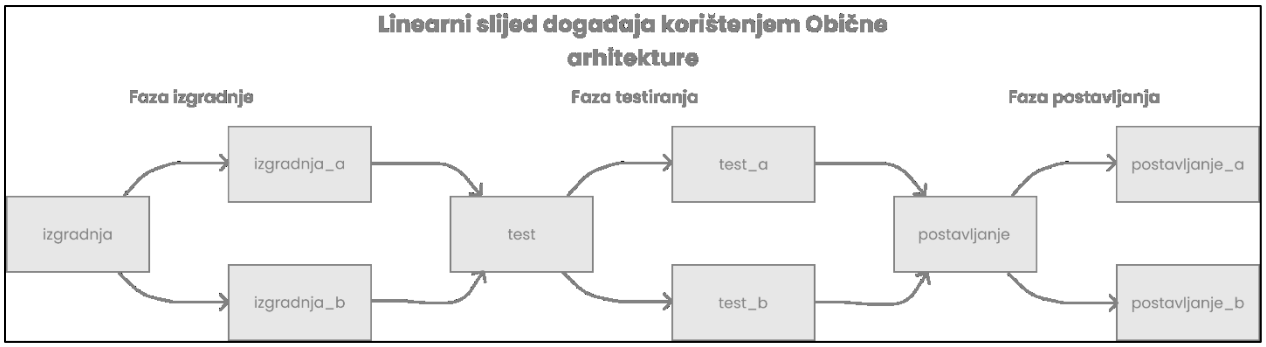
U prikazanom primjeru na slici 26 *rules* element zamijenjen je *only* elementom te ispunjava jednaku svrhu kao i *rules* element ranije s razlikom u tome što nije definiran uvjet da se sama aplikacija mora izgraditi budući da se taj uvjet može ispuniti u zadatku prije.[133] Iako uvjet pokretanja aplikacije nije postavljen, *only* i dalje definira da se zadatak pokreće samo na grani naziva *main* kao i *rules* element ranije. Uz navedene elemente za upravljanje uvjetima pokretanja pojedinačnog zadatka, postoji još i element *needs*. Njegova svrha je definirati ovisnost pojedinog zadatka o drugom zadatku. Zadatak koji ima definiran element *needs* pokrenut će se samo ukoliko zadatak o kojem ovisi uspije i pokrenut će se kad on uspije.[134] Navedeno se može koristiti prilikom provjere pokrivenosti testova gdje jedan zadatak služi za pokretanje testova, a drugi zadatak koji ovisi o prvom zadatku za provjeru pokrivenosti koda testovima. Budući da se definiranjem uvjeta za pokretanje testova može potpuno promijeniti redoslijed pokretanja zadataka u linearnom slijedu

[133] Usp. Isto (2023-07-27)
[134] Usp. Isto (2023-07-27)

događaja i organizacija, odnosno arhitektura linearnog slijeda događaja nije uvijek jednaka te prema tome postoji nekoliko različitih tipova linearnog slijeda događaja.[135]

3.4. Tipovi linearnog slijeda događaja prema arhitekturi i opcije pokretanja linearnog slijeda događaja

Temeljem definicije uvjeta za pokretanje testova, kako je i ranije navedeno, organizacija pokretanja zadataka u linearnom slijedu događaja može varirati i potpuno promijeniti proces linearnog slijeda događaja. Budući da se sam redoslijed izvođenja zadataka može znatno promijeniti uvjetovanjem njihovog samog izvođenja, prema redoslijedu izvođenja zadataka postoji nekoliko različitih arhitektura, odnosno tipova linearnog slijeda događaja.[136] Prvi, ujedno i temeljni oblik arhitekture naziva se Obični linearni slijed događaja.

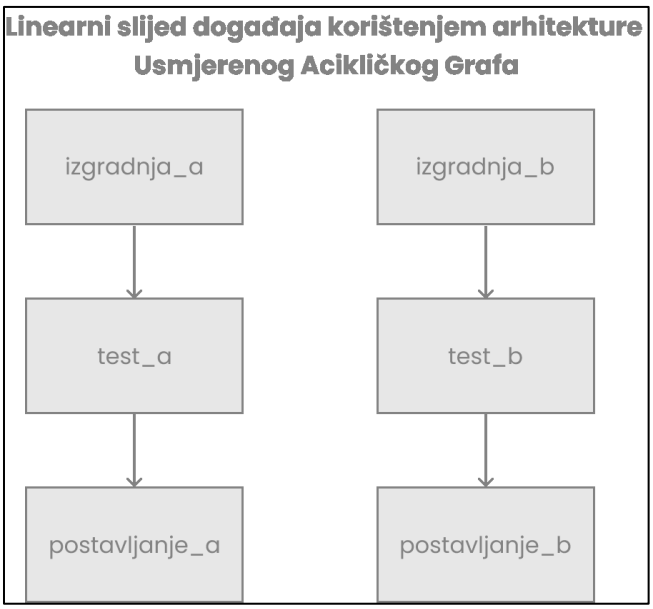


Slika 27. *Prikaz arhitekture Običnog linearnog slijeda događaja*

Ujedno je i najjednostavniji linearni slijed događaja, što se može i zaključiti iz imena, prema njegovoj arhitektu, zadaci se pokreću redom kojim su postavljeni u konfiguracijskoj datoteci, što se može vidjeti i na slici 27 svaka faza, odnosno svi zadaci u pojedinačnoj fazi pokreću se tek onda kad se zadaci u prethodnoj fazi izvrše.[137] Takav oblik arhitekture linearnog slijeda nije najefikasniji i u slučajevima kad se linearni slijed događaja sastoji od puno zadataka i fazi može biti jako kompleksan, ali zbog svoje jednostavnosti pogodan je za održavanje.[138] Nadalje Usmjereni Aciklički Graf je struktura grafa koja ima široku primjenu u područjima računalne znanosti, matematike i informacijske tehnologije. U kontekstu linearnog slijeda događaja arhitektura

[135] Usp. Isto (2023-07-27)
[136] Usp. CI/CD Pipelines: Types of pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/#types-of-pipelines (2023-08-05)
[137] Usp. Pipeline architecture: Basic pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#basic-pipelines (2023-08-05)
[138] Usp. Isto (2023-08-05)

Usmjerenog Acikličkog Grafa odnosi se uspostavljanje veza između zadataka iz različitih faza na način da se izvršavanje obavlja na najbrži mogući način, bez obzira na to kako su faze postavljene.[139]

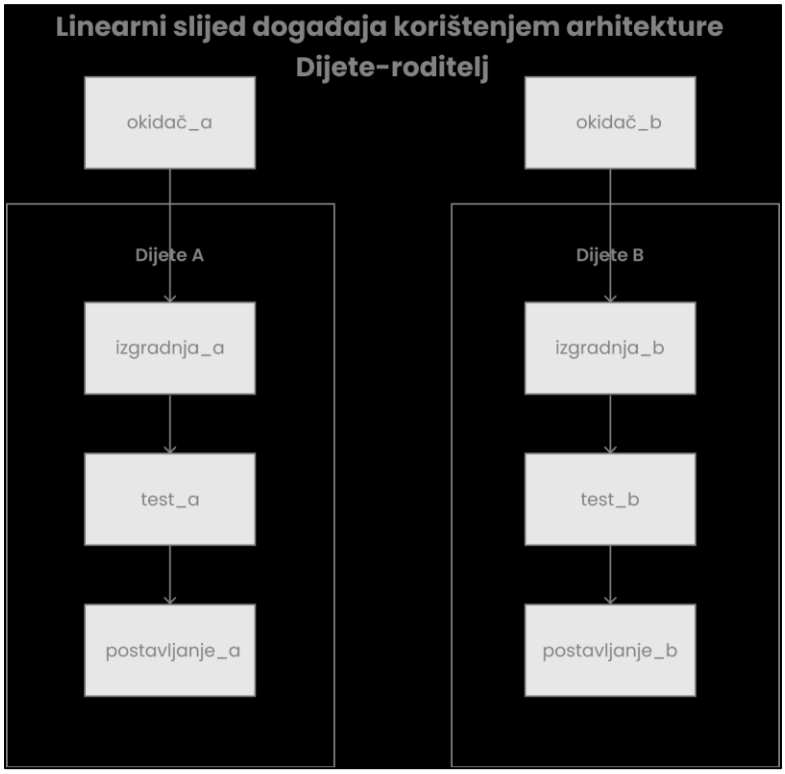


Slika 28. *Prikaz arhitekture* Usmjerenog Acikličkog Grafa

Na direktnom primjeru iz slike 28, brzina izvođenja na najbrži mogući način, odnosi se na činjenicu da ako se zadatak naziva *izgradnja_a,* iz faze *build,* izvrši prije *izgradnja_b zadataka, test_a zadatak* iz faze *test* započet će izvođenje kad sam zadatak *izgradnja_a* završi, odnosno neće čekati da se svi zadaci iz faze *izgradnja* izvrše. Navedeno ponašanje postiže se uvjetovanjem da se zadatak pokrene čim se zadatak o kojem ovisi uspješno izvrši, a direktno u konfiguraciji elementom *needs,* što je prikazano i u ranijem poglavlju vezanom za opcije pokretanja i grupiranja zadataka.[140] Time iako je efikasniji i brži od Običnog linearnog slijeda događaja linearni slijed događaja usmjerenog acikličkog grafa kompleksniji je za održavanje i zahtjeva bolje poznavanje elemenata. Iako arhitektura Usmjerenog Acikličkog Grafa rješava problem efikasnosti i brzo izvodi zadatke u linearnom slijedu događaja koji se sastoji od manjeg broja faza i zadataka, ona nailazi na ograničenja u kompleksnijim sljedovima događaja koji imaju veći broj faza i zadataka pojedinačnim fazama.[141] Zbog prevelike količine zadataka u jednoj fazi i međusobnim ovisnostima

[139] Usp. Pipeline architecture: Directed Acyclic Graph Pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#directed-acyclic-graph-pipelines (2023-08-05)
[140] Usp. Isto (2023-08-05)
[141] Usp. Isto (2023-08-05)

sam linearni slijed događaja postaje manje efikasan te konfiguracijska datoteka teže održiva i graf manje čitljiv. Za rješenje navedenih problema, postoji mogućnost izrade više konfiguracijskih datoteka koje su međusobno povezane i pokreću se na okidanje određenog događaja u linearnom slijedu događaja te se povezuju u jedan. Takav oblik arhitekture naziva za Dijete-roditelj.[142]



Slika 29. *Prikaz arhitekture Dijete-roditelj*

Radi postizanja čitljivosti i smanjenja kompleksnosti pojedinačne konfiguracijske datoteke, kreira se takozvana roditelj konfiguracijska datoteka, koja u sebi može sadržavati standardne zadatke i faze koje služe za izvršavanje određenih radnji nad kodom aplikacije, ali i obavezno sadrži zadatke okidače. Svrha zadatka okidača je pokrenuti drugu konfiguracijsku datoteku koja u sebi sadrži druge faze i zadatke za izvršavanje zadataka koja može biti strukturirana poput ranije navedenih tipova linearnih sljedova događaja.[143] Navedeno je vidljivo i na slici 29, gdje roditeljska konfiguracija sadrži dva zadatka okidač_a i okidač_b koji su okidači te ovisno o njihovom redoslijedu pokreću konfiguracijsku datoteku sa zasebnim zadacima. Okidač_a pokrenut će

[142] Usp. Pipeline architecture: Parent-child pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#parent-child-pipelines (2023-08-05)
[143] Usp. Isto (2023-08-05)

konfiguracijsku datoteku naziva dijete_a te nakon što se izvrše svi poslovi okidač_b pokrenut će konfiguracijsku datoteku dijete_b te su sve navedene konfiguracije dio globalnog linearnog slijeda događaja jedne aplikacije. Time se postiže lakša čitljivost grafa i održavanje konfiguracijskih datoteka jednog linearnog slijeda aplikacije, a uz navedeno smanjuje se i kompleksnost.[144] Pokretanje svih navedenih tipova linearnog slijeda događaja može se, kao i zadaci unutar njih, uvjetovati. Prema definiranju uvjeta pokretanja postoje linearni sljedovi koji se pokreću samo prilikom pokretanja zahtjeva za spajanje Git grana i linearni sljedovi koji se jednako pokreću samo prilikom zahtjeva za spajanje Git grana ali se ponašaju kao da su Git grane već spojene. Svi navedeni tipovi i opcije pokretanja mogu se koristiti prilikom izrade linearnog slijeda događaja za pojedinu aplikaciju, ali važno je uzeti u obzir kompleksnost i svrhu aplikacije i sve kontekste prilikom izrade konfiguracije za linearni slijed događaja.[145]

4. Principi kontinuirane integracije i kontinuiranog postavljanja u konfiguraciji linearnog slijeda događaja

Prikazane opcije konfiguracije linearnog slijeda događaja, definiranje varijabli, fazi i zadataka, upravljanje pokretanjem zadataka i samih linearnih sljedova događaja te sve ostale ranije prikazane opcije konfiguracije omogućuju postavljanje standarda koje kod mora proći prilikom implementiranja principa kontinuirane integracije i kontinuiranog postavljanja. Konfiguracija linearnog slijeda događaja same aplikacije sastoji se od 3 faze: build, test i docker. Prve dvije faze build i test implementiraju principe kontinuirane integracije.

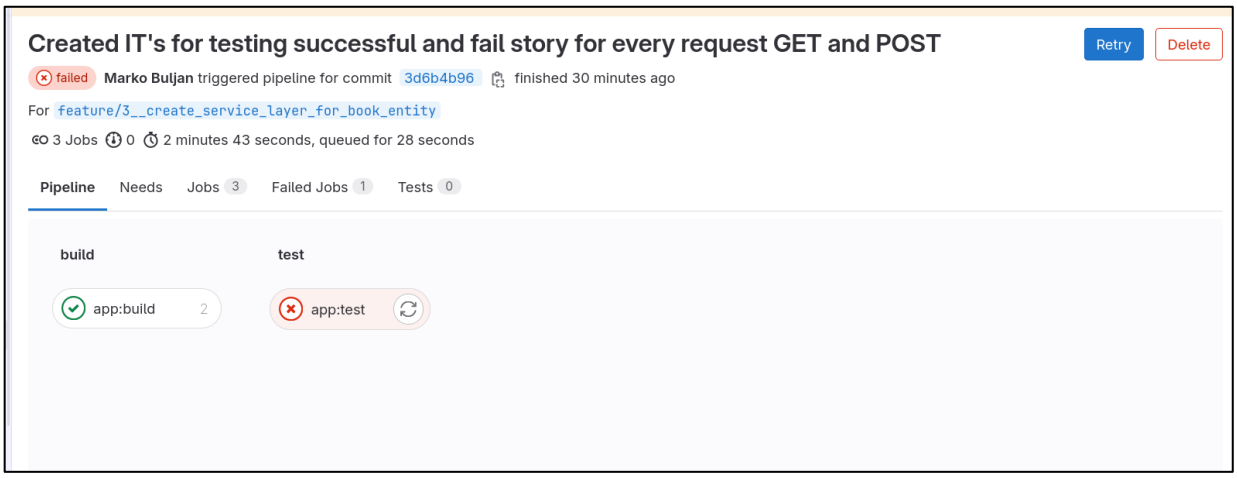
[144] Usp. Isto (2023-08-05)
[145] Usp. Usp. CI/CD Pipelines: Types of pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/#types-of-pipelines (2023-08-05)

Slika 30. Prikaz fazi i zadataka kontinuirane integracije

Kontinuirana integracija, kako je prethodno objašnjeno, praksa je stalne integracije koda, koja se praktično ostvaruje putem skupa automatiziranih skripti za procese izgradnje i testiranja napisanog koda.[146] Konkretno, zadatci naziva app:build 1/2 i app:build 2/2 unutar faze *build* prikazani na slici 30. Napisani su s ciljem izgradnje aplikacije te time provjeravaju uspijeva li napisani kod uopće izgraditi aplikaciju. Osim navedena dva zadataka u fazi build, na slici 30, prikazan je i zadatak nazvan *app:test* zadatak unutar faze test. Zadatak *app:test* pokreće se tek nakon što se zadaci app:build ½ i app:build 2/2 uspješno izvrše. Njegova svrha je izvršiti testove koji su napisani za aplikaciju te će biti uspješan samo ako su se svi testovi uspjeli izvršiti bez problema. Faza testiranja od iznimne je važnosti za svaki linearni slijed događaja budući da omogućuje provjeru ispravnosti, funkcionalnosti i performansi aplikacije, osiguravajući da nema grešaka ili propusta.[147] S obzirom na to da opisani zadaci služe za automatsko izvršavanje naredbi izgradnje aplikacije i pokretanje testova razvijenih za istu, a sama praksa kontinuirane integracije odnosi se na set skripti za

[146] Usp. CI/CD Concepts. URL: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration (2023-08-12)
[147] Usp. Isto (2023-08-12)

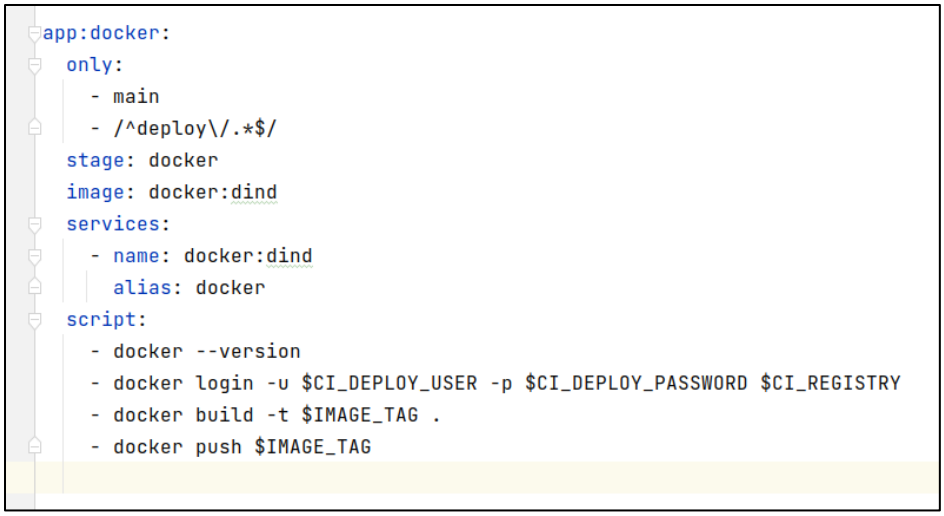
automatizaciju izgradnje i testiranje koda, zadaci potpuno implementiraju praksu kontinuirane integracije. Proces kontinuirane integracije omogućuje da se promjene u kodu brzo i pouzdano integriraju te da se osigura konzistentnost, kvaliteta i stabilnost aplikacije tijekom cijelog razvojnog procesa.[148] Na samom demonstrativnom projektau *book-vinyl-library* navedene faze pokrenut će se neovisno o grani, a kako bi se prikazalo njihovo pokretanje i sama praktična upotreba postavljen je zahtjev da se unutar aplikacije napiše kod koji omogućava izradu i iščitavanje knjiga. Prema već opisanom Git procesum nakon što je kod postavljen na granu, za navedeni zadatak, na repozitoriju, pokrenut je i linearni slijed događaja koji provjerava uspijeva li kod izgraditi aplikaciju i prolazi li testove.



Slika 31. Prikaz neuspješnog linearnog slijeda događaja

Na slici 31 prikazano je grafičko sučelje GitLab platforme koje ilustrira kronološki slijed događaja i faza unutar tog procesa. Kroz jasno definirane grafičke ikonice, primjećuje se da je kod koji je unesen u repozitorij uspješno prošao prvu fazu - fazu izgradnje aplikacije. Međutim, druga faza, koja uključuje testiranje, nije bila uspješna za isti kod. Kod na sporednoj grani zaduženoj za zaseban zadatak ne smije biti spojen s glavnom granom dok sve faze uspješno ne prođu kako bi se na glavnoj grani držao samo onaj kod koji uspješno pokreće aplikaciju. Time se osigurava kvaliteta koda na glavnoj grani što je važno budući da se na glavnoj grani uz prikazane dvije faze koje su zadužene za izgradnju aplikacije i pokretanje napisanih testova za kod aplikacije, pokreće još i faza naziva *docker*.

[148] Usp. Isto (2023-08-12)



Slika 32. Prikaz faze i zadataka kontinuiranog postavljanja

Sam prikazani zadatak app:docker u fazi docker, implementira principe kontinuiranog postavljanja, a prikazano slikom 32 ukratko, ima za cilj izgraditi Docker sliku iz koda i Dockerfile-a. Docker, kao što je već navedeno, odnosi se na program koji omogućava upotrebu i izradu Linux kontejnera čija je definicija objašnjena u ranijem poglavlju 3.2. Gitlab Runner i Runner Executor.[149] U osnovi, Docker slika je svojevrsni predložak koji sadrži sve potrebno za izgradnju i pokretanje kontejnera. Kontejner, s druge strane, predstavlja izolirano izvršno okruženje koje se temelji na određenoj Docker slici. Slike se koriste za stvaranje kontejnera, a kontejneri su instance izvršavanja aplikacije unutar tih slika. Uz sve navedeno važno je naglasiti da Docker slika u sebi sadrži sve konfiguracijske komponente potrebne za izgradnju Docker Kontejner-a.[150] Zadatak *app:docker,* navedeni cilj izgradnje Docker slike postiže kroz naredbe definirane u *script* elementu. Naredbom docker build –t $IMAGE_TAG izrađuje se Docker slika aplikacije s odabranim tagom $IMAGE_TAG, koji služi za jedinstveno identificiranje slike, a naredba docker push –t $IMAGE_TAG šalje sliku s već odabranim tagom u Docker registar. Iz Docker registra dalje se sama Docker slika aplikacije može postaviti na server i omogućiti dostupnom na određenoj adresi. Budući da time zadatak *app:docker* ima svrhu izradu Docker slike aplikacije koja se koristi za dostavljanje aplikacije krajnjim korisnicima, a kontinuirano dostavljanje praksa je dostavljanja

[149] Usp. Schmitt, Jacob. Docker image vs container: What are the differences?. URL: https://circleci.com/blog/docker-image-vs-container/#:~:text=A%20Docker%20image%20executes%20code,own%20unique%20data%20and%20state. (2023-08-12)
[150] Usp. Isto (2023-08-12)

promjena na kodu krajnjim korisnicima, zadatak *app:docker* postiže krajnji cilj kontinuiranog postavljanja i uklapa se u sve principe navedene prakse.[151] Nakon spajanja sporedne grane s kodom koji omogućava izradu i iščitavanje knjiga na glavnu granu, uz faze build i test pokrenuta je i faza docker, što se može vidjeti na slici 33 te se time automatski generirala Docker slika aplikacije s kodom s glavne grane. Time cjelokupna konfiguracijska datoteka definiranim zadacima unutar faza postiže sve principe kontinuirane integracije i kontinuiranog postavljanja i osigurava automatsku kontrolu provjere valjanosti koda, ali i automatizaciju dostavljanja same aplikacije do krajnjih korisnika.[152]

[151] Usp. CI/CD Concepts. URL: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration (2023-08-12)
[152] Usp. Isto (2023-08-12)

5. Zaključak

Praksa Linearnog slijeda događaja, koja se može ukomponirati u rad aplikacijama pomoću alata: Gitlab CI/CD, Github Actions, Jenkins, TeamCity, CircleCI, Bamboo i mnogih drugih, predstavlja nezaobilaznu podršku za programere u današnjem dinamičnom okruženju razvoja aplikacija. Sama implementacija kontinuirane integracije, konkretno u radu Gitlab CI/CD alatom, pomoću prikazanih skripti i faza omogućuje provjeru može li napisani kod pokrenuti aplikaciju, prolazi li kod uspješno napisane testove, ali naravno i sve druge zahtjeve koji se mogu postaviti. Osim kontinuirane integracije, kontinuirano postavljanje, skriptama i fazama putem Gitlab CI/CD-a dodatno pojednostavljuje postupak implementacije aplikacija, čime se štedi vrijeme i osigurava brza dostava novih funkcionalnosti krajnjim korisnicima. Automatizirane skripte i alati za kontinuirano postavljanje omogućuju programerima da se usmjere na razvoj aplikacije, dok se postavljanje i implementacija provode automatski s glavne grane te je zbog toga i nužno da sam kod na glavnoj grani, koji je provjeren skriptama za kontinuirano integraciju, doista bude i valjan. Zbog svega navedenog Gitlab CI/CD, ali i svi ostali alati ne samo da olakšavaju proces izrade aplikacija, već postavljaju temelje standarda kvalitete i pouzdanosti aplikacije. Time raznolika konfiguracijska prilagodljivost, omogućuje programerima da precizno prilagode procese Kontinuirane integracije i Kontinuirane isporuke i dostavljana, kako bi odgovarali specifičnim zahtjevima projekta. Samim time kombiniranjem praksi i alata Linearnog slijeda događaja sa sustavom verzioniranja koda, koji kontrolira promjene nad kodom i metodologijom rada, koja ovisno o metodologiji postavlja principe organizacije, proces izrade aplikacije postaje organiziraniji, dinamičniji te iznimno učinkovit, a sama aplikacija time kvalitetnija i potencijalno profitabilnija nego bez korištenja navedenih praksi.

5. Literatura

1. Advanced configuration. URL: https://docs.gitlab.com/runner/configuration/advanced-configuration.html (2023-06-20)
2. A quick guide to GitLab CI/CD pipelines. URL: https://about.gitlab.com/blog/2019/07/12/guide-to-ci-cd-pipelines/ (2023-06-15)
3. Best 14 CI/CD Tools You Must Know. URL: https://katalon.com/resources-center/blog/ci-cd-tools (2023-06-15)
4. Best Git Hosting Services. URL: https://nira.com/best-git-hosting-services/ (2023-05-20)
5. Brush, Kate. What is Virtualization. URL: https://www.techtarget.com/searchitoperations/definition/virtualization (2023-20-06)
6. Building an Application with Spring Boot. URL: https://spring.io/guides/gs/spring-boot/ (2023-20-05)
7. Choose when to run jobs. URL: https://docs.gitlab.com/ee/ci/jobs/job_control.html (2023-07-27)
8. CI/CD Concepts. URL: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration (2023-06-15)
9. CI/CD Pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/ (2023-06-15)
10. CI/CD Pipelines: Types of pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/#types-of-pipelines (2023-08-05)
11. Cloud from A to Z: Why static websites?. URL: https://acenet-arc.github.io/cloud_from_a_to_z/why-static-websites/ (2023-10-05)
12. Despa, Valentina. A Brief Guide to GitLab CI Runners and Executors, 2021. URL: https://medium.com/devops-with-valentine/a-brief-guide-to-gitlab-ci-runners-and-executors-a81b9b8bf24e (2023-20-06)
13. Difference between JDK, JRE, and JVM. URL: https://www.javatpoint.com/difference-between-jdk-jre-and-jvm (2023-05-12)

1.1 Getting Started - About Version Control. URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (2023-05-12)

14. Gitflow-Workflow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (2023-12-05)
15. GitLab CI/CD variables. URL: https://docs.gitlab.com/ee/ci/variables/ (2023-07-14)

16. Gitlab Runner. URL: https://docs.gitlab.com/runner/ (2023-06-15)
17. IaaS vs. PaaS vs. SaaS. URL: https://www.ibm.com/topics/iaas-paas-saas (2023-05-08)
18. Increase in high-speed internet coverage in 2021. URL: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20220822-1#:~:text=In%202021%2C%2070%25%20of%20EU,to%2037%25%20across%20the%20EU. (2023-10-05)
19. Install Gitlab Runner. URL: https://docs.gitlab.com/runner/install/index.html (2023-06-20)
20. Install GitLab Runner manually on GNU/Linux. URL: https://docs.gitlab.com/runner/install/linux-manually.html (2023-06-20)
21. IntelliJ IDEA overview. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (2023-05-20)
22. Introduction to the Spring Framework: Part I. Overview of Spring Framework. URL: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html (2023-05-15)
23. Issue boards. URL: https://docs.gitlab.com/ee/user/project/issue_board.html (2023-05-24)
24. Javadoc Tool. URL: https://www.oracle.com/java/technologies/javase/javadoc-tool.html (2023-07-27)
25. Jobs. URL: https://docs.gitlab.com/ee/ci/jobs/ (2023-07-14)
26. Jobs: Group jobs in a pipeline. URL: https://docs.gitlab.com/ee/ci/jobs/#group-jobs-in-a-pipeline (2023-07-27)
27. Kanban – Agile Methodology. URL: https://www.geeksforgeeks.org/kanban-agile-methodology/ (2023-05-23)
28. Kubernetes Components. URL: https://kubernetes.io/docs/concepts/overview/components/ (2023-06-20)
29. Leiner, B.M., Cerf, V.G., Clark, D.D., et al. A Brief History of the Internet. ACM SIGCOMM Computer Communication Review, 39, 22-31 str. URL: https://doi.org/10.1145/1629607.1629613 (2023-10-05)

30. Lushpenko, Maksym. Creating Docker config.json for external systems. URL: https://lumaks.medium.com/creating-docker-config-json-for-use-in-external-systems-a5d14aa81c41 (2023-06-25)
31. Metodologija. URL: https://www.enciklopedija.hr/natuknica.aspx?id=40441 (2023-05-08)
32. Migrating to the new runner registration workflow. URL: https://docs.gitlab.com/ee/ci/runners/new_creation_workflow.html (2023-07-03)
33. Pipeline. URL: https://www.techopedia.com/definition/5312/pipeline (2023-06-15)
34. Pipeline architecture: Basic pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#basic-pipelines (2023-08-05)
35. Pipeline architecture: Directed Acyclic Graph Pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#directed-acyclic-graph-pipelines (2023-08-05)
36. Pipeline architecture: Parent-child pipelines. URL: https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html#parent-child-pipelines (2023-08-05)
37. Repository. URL: https://docs.gitlab.com/ee/user/project/repository/ (2023-05-20
38. Runner SaaS. URL: https://docs.gitlab.com/ee/ci/runners/index.html
Schmitt, Jacob. Docker image vs container: What are the differences?. URL: https://circleci.com/blog/docker-image-vs-container/#:~:text=A%20Docker%20image%20executes%20code,own%20unique%20data%20and%20state. (2023-06-20)
39. Semantika. URL: https://enciklopedija.hr/natuknica.aspx?ID=55330 (2023-06-15)
40. Singleton in Java. URL: https://refactoring.guru/design-patterns/singleton/java/example (2023-13-05)
41. Spring Framework – Overview. URL: https://www.tutorialspoint.com/spring/spring_overview.htm (2023-12-05)
42. 11 Top Most Popular Software Development Methodologies. URL: https://www.spaceo.ca/blog/software-development-methodologies/ (2023-05-10)

43. Tutorial: Create your first Spring application.
URL: https://www.jetbrains.com/help/idea/your-first-spring-application.html (2023-05-20)
44. Version Control System. URL: https://www.geeksforgeeks.org/version-control-systems/ (2023-05-12)
45. What is CI/CD. U https://www.redhat.com/en/topics/devops/what-is-ci-cd (2023-06-15)
46. What is Java ? URL: https://www.javatpoint.com/java-tutorial (2023-05-12)
47. What's a Linux container?. URL: https://www.redhat.com/en/topics/containers/whats-a-linux-container (2023-06-20)
48. What is the Linux kernel? URL: https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel (2023-06-20)
49. What is Maven?. URL: https://maven.apache.org/what-is-maven.html (2023-07-23)
50. What is YAML?. URL: https://www.redhat.com/en/topics/automation/what-is-yaml (2023-07-23)
51. Yakutovich, Anton. GitLab CI: Cache and Artifacts explained by example. URL: https://dev.to/drakulavich/gitlab-ci-cache-and-artifacts-explained-by-example-2opi (2023-07-23)
52. Yesmin, Fahmida. Bash base64 encode and decode. URL: https://linuxhint.com/bash_base64_encode_decode/ (2023-06-25)