Зборник радова Факултета техничких наука, Нови Сад

UDK: 004.9
DOI: https://doi.org/10.24867/07BE34Lalic

ЖИВОТНИ ЦИКЛУС ВЕБ АПЛИКАЦИЈА КАО СОФТВЕРСКИХ ПРОИЗВОДА УЗ ОСЛОНАЦ НА АЛАТЕ ЗА CI/CD

LIFE CYCLE OF WEB APPLICATIONS AS SOFTWARE PRODUCTS BASED ON CI/CD TOOLS

Милена Лалић, Бранко Милосављевић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У раду су анализиране савремене методологије развоја софтвера које укључују и концепте непрекидне интеграције и непрекидне експлоатације. Анализирани су алати који помажу развој веб апликација у складу са овим концептима. У складу са предложеном методологијом је имплементирана веб апликација за резервацију стола у ресторану.*

Кључне речи: *непрекидна интеграција, непрекидна експлоатација*

Abstract – *The paper analyzes the modern methodologies of software development that include concepts of continuous integration and continuous delivery. Tools that improve web application development with these concepts are analyzed. Following the proposed methodology, a web-based restaurant reservation application was implemented.*

Keywords: *continuous integration, continuous delivery*

1. УВОД

Continuous Integration (CI), *Continuous Delivery* (CD) и *Continuous Deployment* (CD) се једним именом називају континуалне праксе које имају за циљ убрзање развоја и испоруке софтверских производа, без смањења квалитета.

Овај рад има за циљ истраживање аутоматизације и усвајање континуалних пракси и алата који омогућавају њену реализацију. Резултати који се добију могу се користити као смернице за повећање свести примене одговарајуће праксе.

2. КОНТИНУАЛНЕ ПРАКСЕ

2.1. Континуална интеграција

2.1.1. „Мрачно“ доба интеграције

Ово поглавље описује мануелни процес и њему својствене недостатке. Описан је пример, који осликава поглед сваког од чланова тима (програмери, тестери и *project manager*).

2.1.2. Шта је континуална интеграција?

Континуална интеграција је пракса која помаже програмерима да испоруче бољи софтвер на поузданији начин. Главна идеја је поделити велике целине, које се развијају, на мање. Битно је да се интеграциони догађају дешавају често. Идеалан случај је да се ово дешава неколико пута на дан, а све то у циљу раног откривања грешака. И последња, али ништа мање важна ставка је процес аутоматског тестирања сваке промене.

2.1.3. Предности коришћења континуалне интеграције

Смањује ризик.

Смањује мануелне процесе који се понављају.

Генерисање софтверског производа који је спреман за испоруку у било ком тренутку.

Већа сигурност у исправно извршавање софтверског производа од стране развојног тима.

2.1.4. Шта спречава тимове да користе CI?

Повећани трошкови одржавања.

Превише промена.

Превише неуспешних build- ова.

Програмери би требали да обављају ове активности.

2.1.5. Праксе и принципи

Континуална пракса може да се уведе у било ком тренутку, међутим она се не односи само на инсталирање алата и њихово коришћење. Заправо, то је потпуно инжењерска пракса која захтева дисциплину.

Управо из тога произилази да је најбоље увести одмах на почетку пројекта и самим тим навикавати чланове тима на нове технологије и развојни процес.

Што чешће интегрисати измене.

Уколико build не прође, потребно је одмах поправити грешку.

Писање тестова.

Сви тестови морају успешно да се изврше.

Локално покренути build.

Не качити код на репозиторијум система за контролу верзија, уколико претходни build није успешно извршен.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био Бранко Милосављевић, ред. проф.

2.2. Континуална пракса

2.2.1. „Мрачно“ доба испоруке

Извођење процеса испоруке је био догађај „великог праска“. Након што се сматрало да је апликација покривена тестовима, неко из тима би добио задатак да запакује апликацију и да је испоручи. Испорука је, такође, веома стресна фаза у циклусу развоја и традиционални приступ је укључивао много мануелних корака. Испорука се дешавала веома ретко, али и дан данас постоје компаније које једном у шест месеци испоруче нову верзију.

2.2.2. Шта је континуална испорука?

Континуална испорука је пракса паковања и припреме апликације за продукцију, што је чешће могуће. Континуална испорука води континуалну интеграцију корак даље. Свака нова функционалност је потенцијални кандидат за пуштање у продукцију. Опет зависно од организације, али у суштини да ли ће одређена функционалност ићи у продукцију или неће, захтева људску интервенцију. Једном када је CD постављен, процес испоруке постаје тривијалан, јер се све може обавити притиском једног дугмета.

2.3. *Continuous deployment*

Овај развојни приступ надограђује континуалну испоруку и у потпуности уклања људску интервенцију. Сваки кандидат за *release* за који се сматра да је спреман (тј. сви тестови пролазе) се аутоматски пушта у продукцију.

2.3.1. *Continuous delivery* и *continuous deployment*

Continuous delivery обухвата низ пракси које обезбеђују да се имплементација софтверског производа може пустити у продукцију на брз и сигуран начин тако што ће се вршити испорука сваке измене у окружење које је слично продукцијском. Пуштање измена у продукцију се врши притиском на једно дугме.

Continuous deployment је следећи корак након *continuous delivery* фазе. Свака измена која прође аутоматизоване тестове се аутоматски пушта у продукцију.

Поента је, на основу пословних потреба, одлучити да ли је *continuous deployment* управо оно што је компанији неопходно. Насупрот томе, *continuous delivery* је апсолутно неопходна, јер све измене могу да се испоруче клијенту кликом на једно дугме.

3. АЛАТИ ИМПЛЕМЕНТИРАНОГ РЕШЕЊА

3.1. Систем за контролу верзија

Примена континуалне интеграције је незамислива без система за контролу верзија. Заправо, тешко је замислити процес израде софтверског производа без њега.

Систем за контролу верзија управља променама над дигиталним артифактима (најчешће фајловима и фолдерима). Постоје две архитектуре оваквих система. Код централизованих система сва историја се налази у репозиторијуму централног сервера, а клијенту имају само текућу верзију.

Представници су *SVN, Perforce, Subversion.* Код дистрибуиране архитектуре, клијент има увид и пуну историју измена. Представници су *Git, Mercurial* и *Bazaar.*

3.1.1. *Git*

Git је дистрибуирани систем за контролу верзија. *Git* о подацима размишља као о скупу снимака (*snapshot*- а) минијатурног фајл система.

Репозиторијум је скуп метаподатака.

Радни директоријум је локални фолдер у коме се мењају и креирају фајлови који су обухваћени верзионирањем.

Простор припреме фајл који чува информације о томе шта ће бити део следећег трајног бележења.

Удаљени репозиторијум је место где се чувају фајлови пројеката на неком серверу.

HEAD је замишљени показивач.

Команда add додаје фајл у локални репозиторијум и припрема га за *commit.*

Команда commit прави снимак промена стања фајлова у односу на претходно.

Командом push се шаљу измене из локалног репозиторијума на удаљени репозиторијум.

Командом fetch се повлачи последње стање са удаљеног репозиторијума.

Грана је алтернативни ток развоја.

3.1.2. *GitLab*

GitLab је систем за управљање *git* репозиторијумима. Софтвер је креирао *Dmitriy Zaporozhets* 2011. године. Првобитно је написан у програмском језику *Ruby*, а касније су неки делови пребачени у програмски језик *Go*.

3.2. *Jenkins*

Jenkins је сервер отвореног кода континуалне интеграције који оркестрира низом акција на аутоматизован начин. Предност у односу на друге алате је та што нуди велики број додатака.

3.2.1. Основни појмови *Jenkins* сервера

Jenkins Pipeline је група задатака који су међусобно повезани. Задаци се извршавају један за другим, односно након успешно извршеног једног задатка прелази се на други.

Корак (step) је основни део *pipeline*- а и говори *Jenkins*- у шта да уради.

Jenkinsfile је текстуални фајл у ком се врши дефинисање *pipeline*- а.

Чвор (node) је основни концепт *scripted pipeline*- а.

3.3. *Sonatype Nexus*

Nexus је менаџер складишта. Омогућава посредан приступ централном репозиторијуму, прикупљање и управљање зависностима.

Централни репозиторијум је скраћено од *Central Maven Repository (CMR)* и може се посматрати као глобални менаџер репозиторијума који чува све компоненте отвореног кода.

3.4. *Docker*

Docker је алат отвореног кода који је дизајниран да олакша креирање, *deploy* и покретање апликације. Заснован је на идеји паковања кода апликације са

свим зависностима у преносиву јединицу која се зове контејнер. Захваљујући томе, програмери могу бити сигурни да ће апликација радити на било којој машини.

3.4.1. *Docker* слика

Docker слика је фајл, који се састоји од више слојева. Контејнер се стартује на основу слике. Слике се чувају у *Docker* регистру, као што је *DockerHub* и могу се преузети *docker pull* командом.

3.4.2. *Dockerfile*

Dockerfile је текстуални документ који садржи све наредбе које би се обично ручно извршавале да се изградила *Docker* слика. Свака команда ствара један свој слике.

FROM. Прва команда. Дефинише базну слику.

EXPOSE. Отвара задати порт.

RUN. Омогућава инсталацију апликација и пакета који су неопходни.

ENTRYPOINT. Омогућава конфигурацију контејнера.

4. ИМПЛЕМЕНТИРАНО РЕШЕЊЕ

4.1. *NoWait* апликација

Пројектни задатак представља апликација која омогућава корисницима да резервишу сто у жељеном ресторану у одређеном термину. Резервација је омогућена у различитим ресторанима који су регистровани у оквиру система.

4.2. Процес рада имплементираног решења

Програмери, локално на својим машинама пишу код за одређену функционалност. Оног тренутка када програмери покрену команду *push* система за контролу верзија, сав код се чува на репозиторијуму. На *GitLab*- у је подешен *webhook* који ће обавестити *Jenkins* да се десила измена. *Jenkins* на основу *Jenkinsfile*- а извршава задатке.

5. ЗАКЉУЧАК

Овај рад представља увод у нове термине и праксе које пре или касније чекају развој софтверских производа. Постављена архитектура обезбеђује израду континуалних пракси, које својом толеранцијом на промене, усмеравају и контролишу развојни процес. Дато решење обезбеђује рано откривање грешака, њихово брзо уклањање и побољшање квалитета.

Недостаци који захтевају будућа истраживања:

Vagrant – алат за брзо и једноставно подешавање комплетног *development* окружења виртуелне машине. Једноставним начином за коришћење и фокусом на аутоматизацију, смањује време за подешавање радног окружења.

Kubernetes - систем за покретање и координацију контејнерских апликација. Платформа је дизајнирана за потпуно управљање животним циклусом контејнерских апликација и услуга користећи методе које пружају предвидљивост, скалабилност и велику доступност.

6. ЛИТЕРАТУРА

[1] Before CI/CD, https://thenewstack.io/understanding-the-difference-between-ci-and-cd/

[2] Paul Duvall, Steve Matyas, and Andrew Glover, Continuous Integration Improving Software Quality and Reducing Risk,

[3] Lauri Hukkanen, Adopting Continuous Integration – A Case Study, Aalto University, 2015.

[4] Continuous integration vs delivery vs deployment, https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[5] Manoj Mahalingam S, Learning Continuous Integration with TeamCity, Packt Publishing, 2014.

[6] Игор Дејановић, Системи за контролу верзија - увод, слајдови са предавања, Универзитет у Новом Саду, Нови Сад, 2016., http://www.igordejanovic.net/courses/tech/sistemi-za-kontrolu-verzija.html#1

[7] Scott Chacon, Ben Straub, Pro Git, apress, 2014.

[8] Игор Дејановић, *Git DVCS*, слајдови са предавања, Универзитет у Новом Саду, Нови Сад, 2018., http://www.igordejanovic.net/courses/tech/git.html#1

[9] Git pull, https://www.atlassian.com/git/tutorials/syncing/git-pull

[10] About Jenkins, https://www.guru99.com/jenkin-continuous-integration.html

[11] Jenkins Pipeline, https://jenkins.io/doc/book/pipeline/

[12] Syntax Comparison, https://jenkins.io/doc/book/pipeline/syntax/#compare

[13] Sonatype Nexus, https://blog.sonatype.com/2010/04/why-nexus-for-the-non-programmer/

[14] Jeff Nickoloff, Docker in Action, Manning Publications Co., 2016

Биографија

Милена Лалић је рођена 19.6.1994. године у Книну, Хрватска. Основну школу „Михајло Пупин“ је завршила 2009. године у Ветернику. Гимназију „Исидора Секулић“ завршила је у Новом Саду, 2013. године. Исте године је уписала Факултет техничких наука, смер Рачунарство и аутоматика. Основне академске студије завршава 2017. године и исте године уписује мастер академске студије из области Рачунарство и аутоматика, Примењене рачунарске науке и информатика. У септембру 2018. године је положила последњи испит и тиме стекла услов за одбрану мастер рада. Тренутно ради као *software developer* у фирми *Pannonia-Expertise.*