Jenkins ukratko

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Uroš Milenković, Nikola Sojčić, Bojan Nestorović
uros.milenkovic@outlook.com, soja.991@gmail.com, bojants91@gmail.com

13. april 2016.

Sažetak

Danas kada se softver razvija sve većom brzinom, timovi programera sve više rastu, a klijenti sve zahtevniji, potrebno je imati sistem koji integriše raznolike alate koji se koriste od pisanja prve linije kôda do finalnog proizvoda koji se isporučuje klijentu. Jedan od najpopularnijih sistema za integraciju softvera je Jenkins. On automatizuje procese build-ovanja, testiranja, i depoloyment-a. U ovom radu pokušaćemo da "zagrebemo" površinu ovog složenog sistema i u kratkim crtama opišemo postupak korišćenja.

Sadržaj

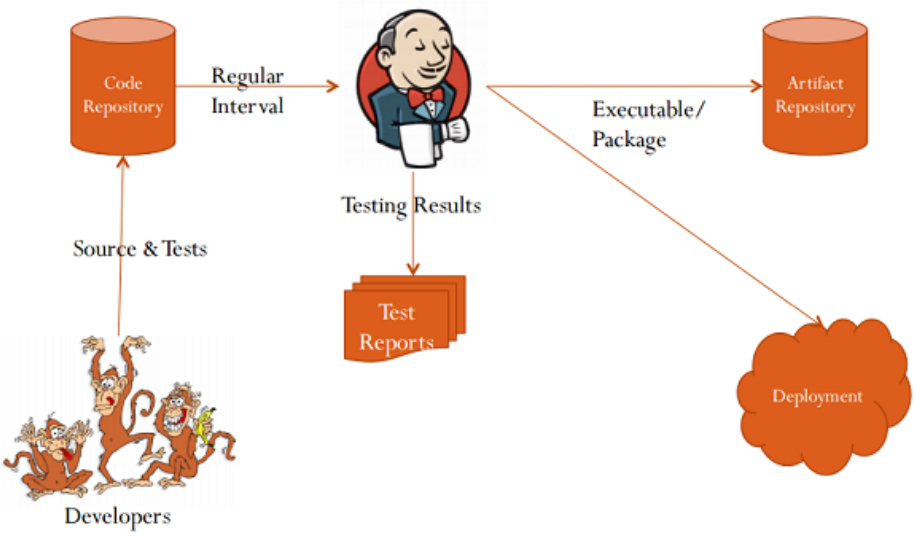
1 Uvod

Jenkins je softverski alat za kontinuiranu integraciju softvera (eng. *Continuous software integration*) napisan na Java programskom jeziku. Jenkins omogucava timovima da se fokusiraju na posao tako što automatizuje proces build-ovanja, testiranja i objavljivanja novih verzija softvera. Ono što jenkins izdvaja je to što postoji bogata baza dodataka (eng. *Plugins*), lako razvijanje sopstvenih dodataka i podržan je od strane velike zajednice. Originalno je razvijan kao Hudson projekat u okviru Sun Microsistems-a. Krajem 2010-te godine došlo je do nesuglasica između originalnih razvijaoca i Oracla koja su rezultirala izdvajanja projekta i promenom imena iz Hudson u Jenkins. Oracle je nastavio da razvija Hudson i smatra Jenkins kao poseban projekat, a ne kao promenu imena.

Kontinuirana integracija predstavlja način razvijanja softvera gde članovi tima objedinjuju svoj kod na dnevnom nivou. Svako objedinjavanje je potvrđeno od strane automatskog build-ovanja i testiranja kako bi se detektovale greške što je brže moguće. Svaka promena mora proći kroz sve korake validacije na svom putu ka objavljivanju. Ovakav pristup smanjuje cenu projekta, vreme uloženo i rizik pri konstantnim objavama novih verzija.

Prilikom svakog objedinjavanja, sistem je:

- Integrisan - sve promene do tog trenutka su objedinjene u projekat
- Build-ovan - kod je kompajliran u paket ili u izvršni fajl
- Testiran - pokreću se automatski testovi
- Arhiviran - verzionisan i sačuvan kako bi se distribuirano(ako se to želi)
- Primenjen - učitan na sistem gde developeri mogu da interaktuju sa najnovijim promenama



Slika 1: Kontinuirana integracija softvera

2 Build Jobs

Build jobs predstavlja skup poslova u koji spadaju kompajliranje, testiranje, pakovanje, razvijanje projekta ili bilo koji drugi posao koji manevriše sa vašim projektom. Build job je osnovni pojam kad se govori o principu kontinualne integracije.

2.1 Kreiranje Build Job-a

Pravljenje build job-a je veoma jednostavno i ono podrazumeva razna podešavanja. Prvo treba odrediti kakvog će tipa biti projekat. Postoje četiri osnovne vrste i to su:

- Slobodan projekat(Freestyle software project) - Vrsta projekta koji pružaju maksimalnu fleksibilnost i koji služe za osnovnu upotrebu
- Maven project - buiild job koji je specijalno namenjen Maven projektima
- External job - ova opcija služi da se prate izvršavanja nekog drugog procesa koji se ne nalazi na Jenkins-u
- Multi-configuration project - ova opcija je pogodna za projekte koji zahtevaju više različitih konfigurisanja
- Copy existing job - projekat može da bude i kopija već postojećeg projekta koji zahteva neke promene u konfigurisanju

2.1.1 Kreiranje slobodnog projekta

Slobodan projekat je najfleksibilnija vrsta i može se koristiti za bilo koju vrstu projekta. Puno opcija koje se nameštaju u okviru slobodnog projekta se javljaju i u ostalim vrstama projekta. Pri kreiranju projekta prvo se unose osnovni podaci kao što su ime projekta i opis projekta. Zatim slede opcije:

- Discard Old Builds - čekiranjem ove opcije limitirate broj build-ova koji će se čuvati, postoje dva kriterijuma:
  - Po starosti - build se briše posle određenog vremena
  - Po broju - čuva se N build-ova
- This build is parameterized - korisnik dodeljuje parametre koje će build koristiti. Neki od parametara su:
  - Boolean Parameter - definiše se string ”true” ili ”false” koji se može koristiti u procesu build-ovanja
  - Choice Parameter - definiše se string koji može imati bilo koju vrednost iz liste izbora i koji se može koristiti u procesu build-ovanja
  - Password Parameter - definiše se tekst gde korisnici mogu da unesu string vrednost koji se može koristiti tokom proces build-ovanja
- Disable Build - kad je čekirano privremeno se obustavlja build-ovanje
- Execute concurrent builds if necessary - čekiranjem ove opcije, omogućava se paralelno izvršavanje build-ovanja. Veoma kosrisno kod parametrizovanih build-ova gde su izvršavanja nezavisna jedna od drugih

”Napredne” opcije:

- Quiet period - čekiranjem ove opcije, zadaje se broj sekundi nakon koliko će se pokrenuti sledeći build
- Retry Count - u slučaju da build ne uspe, zadaje se broj ponovnih pokušaja
- Block build when upstream project is building - sprečava se build-ovanje ako je neki projekat koji je zavistan od njega isto u procesu build-ovanja
- Block build when downstream project is building - sprečava se build-ovanje ako je neki projekat koji je "dete" isto u procesu build-ovanja
- Use custom workspace - manualno zadavanje radnog prostora
- Keep the build logs of dependencies

2.2 Integracija sa izvornim kodom

Jedna od najbitnijih i najznačajnijih stvari je integracija sa sistemom za kontrolu verzija. Jenkins prati svaku promenu u vašem izvornom kodu posle kojih se pokreće kompajliranje i razni automatski testovi. Podržani su razni sistemi za kontrolu verzija, CVS i Subversion "u startu", dok za sisteme kao što su Git, Mercurial, Harvest, BitKeeper i ostale postoje plugin-ovi koji se lako instaliraju preko Jenkins plugin Manager-a. Pri pravljenju projekta i odabiru koji sistem za kontrolu verzija će se koristiti, unošenjem URL-a repozitorijuma se vrši integracija sa izvornim kodom.

2.3 Git Setup

Da bismo mogli da se povežemo sa Git-om prvo moramo instalirati Git. Za operativne sisteme Windows i Mac OS postoje instalacije, a na Linuxu se instalira putem jednostavne komande:

```
sudo apt-get install git-all
```

Za razliku od sistema za kontrolu verzija kao što su CVS i Subversion čiji su plugin-ovi već instalirani, plugin za Git, koji se nalazi u Jenkins-ovom plugin Manager-u, morate sami instalirati. Nakon instalacije pri pravljenju novog projekta, u opcijama za biranje sistema za kontrolu verzija otvoriće se opcija i za Git što je prikazano na slici 2

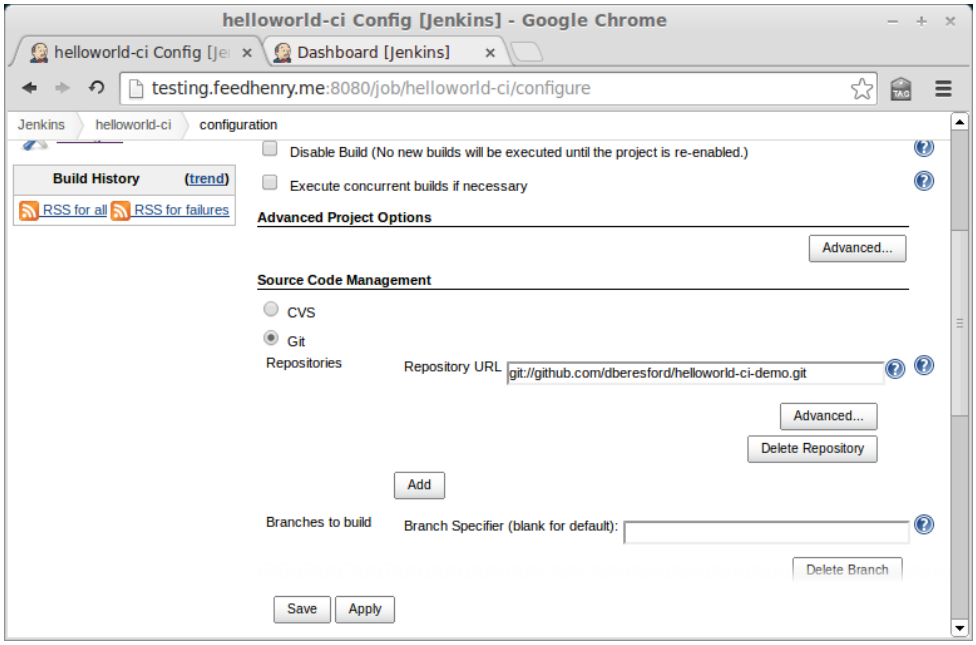
Uz opcije "Repository" gde se navodi URL repozitorijuma i "Branches to build" gde se navodi ime branch-a(grane) koje se build-uje postoje je tzv. "Dodatna ponašanja"(Additional Behaviours). Neka od njih su:

- Polling ignores commits from certain paths
  1. Included regions - unose se putanje fajlova koje se build-uju
  2. Excluded regions - unose se putanje fajlova čije testiranje nema efekta npr. slike
- Polling ignores commits in certain users

  Excluded users - imena user-a od čije strane ne može da se pokrene build-ovanje

2.4 Pokretanje build-ova

Nakon nameštanja koji ćete sistem za kontrolu verzija koristiti, vreme je da se konfiguriše kada će se build-ovi pokretati. Postoje osnovne tri vrste pokretanja build-ova, a to su:



Slika 2: Git

- Build after other projects are built - ova opcija pruža pokretanje build-a kad god se neki drugi build izvrši. Postoje i 3 dodatne opcije:
  - Trigger only if build is stable - pokreni build samo ako je build stabilan
  - Trigger even if the build is unstable - pokreni build iako je build nestabilan
  - Trigger even if the build fails - pokreni build iako build ne uspe
- Build periodically

  Odlika kontinualne integracije jeste kontinualno izvršavanje build-ovanja nakon svake promene što nije stvar kod periodičnog build-ovanja. U nekim slučajevima je i pogodno koristiti ovu vrstu pokretanja npr. kod projekata gde iscrpna testiranja traju i po nekoliko sati i gde je pogodnije pokretati build-ove u određenim vremenskim intervalima. Sintaksa za zadavanje vremenskog intervala:
  MINUTE HOUR DOM MONTH DOW
  - MINUTE - minut u satu (0-59)
  - HOUR - sat u danu (0-23)
  - DOM - dan u mesecu (1-31)
  - MONTH - mesec(1-12)
  - DOW - dan u nedelji(0-7) gde su 0 i 7 nedelje
- Poll SCM

  Bolja strategija od periodičnog build-ovanja jeste Polling the SCM tzv. ”ispitivanje” sistema za kontrolu verzija. Ideja je da se sistem za kontrolu verzija ”ispituje” da li je napravljena izmena u izvornom kodu. U slučaju da jeste Jenkins će pokrenuti build.

2.5 Koraci build-ovanja

Definisanjem build koraka govorite Jenkins-u šta da radi sa vašim izvornim kodom. Jedan build može da ima više koraka u koje spadaju izvršavanje shell skripte, izvršavanje Windows batch komande ili povezivanje sa Maven-om ili Ant-om.

2.6 Akcije posle build-ovanja

Nakon samog procesa build-ovanja potrebno je preduzeti neke akcije. Neke od mogućih akcija su:

- Archive the artifacts - ova opcija omogućava da se rezultati build-ovanja čuvaju u određenom formatu i da se zatim download-uju
- E-mail Notifications - Jenkins obaveštava mejlom svaki put kada build ne uspe, kada uspe build nakon neuspelog pokušaja, kada je build nestabilan nakon uspešnog build-ovanja.

3 Automatsko testiranje

Jedna od bitnih aktivnost u ciklusu je automatsko testiranje softvera [1]. Automatsko testiranje značajno unapređuje razvojni proces. Ovakvi testovi daju sigurnost da nove izmene u kodu neće pogoršati stabilnost sistema, kao i da će funkcionalnosti koje su do tada radile, nastaviti da rade. Timovi treba da ulažu u pisanje testova jer se njima podiže kvalitet proizvoda, a cena testiranja je mnogo manja u odnosu na kasnije ispravke.

Najefikasniji pristup pisanja dobrih testova je da se testovi pišu na početku, pre pisanja samog kôda. Ovakva tehnika se naziva razvoj vođen testovima (eng. *Test Driven Development*). U te svrhe se najčešće koriste testovi jedinica (eng. *Unit tests*).

U ovom poglavlju ćemo pokazati kako se u Jenkins može integrisati automatsko testiranje. Iako postoji mnogo različitih načina za testiranje softvera, držaćemo se uglavnom testova jedinica jer smatramo da je takav pristup testiranju najzastupljeniji.

3.1 Uključivanje testova jedinica u proces

Postoje različiti testovi jedinica za razne jezike. Najpoznatija grupa testova jedinica su xUnit testovi, gde x označava programski jezik (za Java programski jezik će biti JUnit, za PHP će biti PHPUnit itd.).

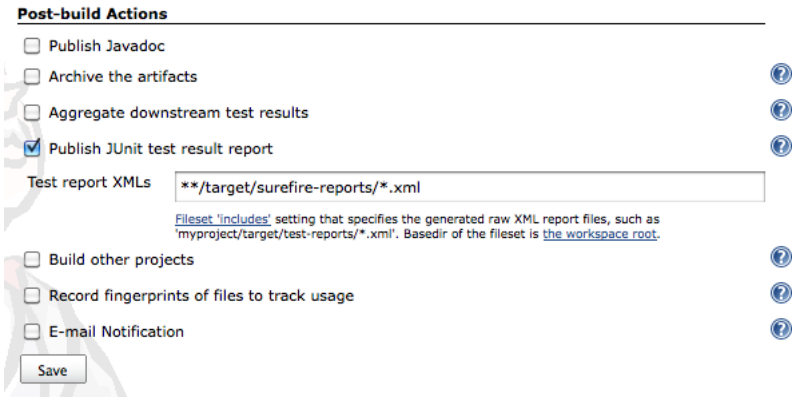
xUnit testovi kao rezultat testiranja generišu neku vrstu izveštaja u XML datoteku za koju se zna shema. To znači dve stvari. Prvo, da alati za testiranje ne moraju da budu baš iz xUnit familije, već mogu biti bilo kakvi alati koji će svoj izveštaj generisati u XML datoteku. Drugo, da Jenkins može da parsira taj izveštaj i lepše da ga prikaže za razliku od XML datoteke.

Automatsko testiranje se izvršava kao jedna od operacija build procesa. Pošto je izlaz testiranja XML datoteka, jedino što preostaje da se uradi je da se konfiguraciji projekta postavi putanja do izlazne XML datoteke.

3.1.1 Konfigurisanje test izveštaja i prikazivanje rezultata

Prvo je potrebno dodati novu post-build akciju. Iz padajućeg menija treba izabrati "Publish JUnit test result report". Jenkins dolazi sa preinstaliranim dodatkom za JUnit test izveštaje. Ako je projekat testiran

u nekom drugom xUnit alatu, potrebno je instalirati plugin koji se zove "xUnit"i njega koristiti. Od konfiguracije build menadžera zavisi putanja XML izveštaja. Tu putanju upisujemo u polje "Test report XMLs", kao što je prikazano na slici 3.



Slika 3: Podešavanje putanje za XML izveštaj

Kada je konfigurisan test izveštaj, Jenkins na početnoj strani projekta prikazuje grafikon na kome se mere za uspešne i neuspešne testove. Kao što se sa slike 4 vidi, uspešni su obojeni plavom bojom, dok su neuspešni obojeni crvenom bojom.

U sekciji sa linkovima na početnoj strani možemo videti poslednji build, poslednji stabilni build i poslednji uspešni build. Izborom linka za poslednji build prelazimo na ekran sa detaljima o tom buildu. Na slici 5 je prikazan ekran sa detaljima. U detaljima Jenkins nam prikazuje koji testovi nisu prosli. Klikom na link neuspešnog testa dobijamo ekran sa detaljima zašto određeni test nije prošao. Ovako se jednostavno fokusiramo na greške i možemo ih brže ispraviti.

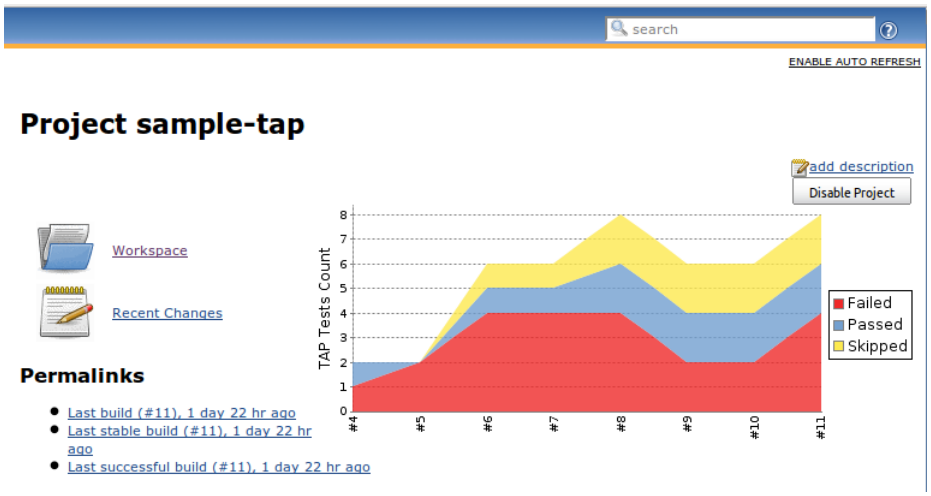
Testovi jedinica su dizajnirani tako da budu brzi. Na većim projektima testovi mogu da traju i po par sati, a nekada se oni pokreću i više puta dnevno. Spori testovi troše dragoceno vreme programera. Zato je veoma bitno da se testovi pišu tako da budu efikasni. U tu svrhu veoma je poželjno da imamo povratnu informaciju o tome koliko testovi dugo traju, kako bismo mogli da uvidimo problem. Srećom, Jenkins nam pruža lepu vizuelizaciju koliko dugo su se testovi izvršavali za svaki build. Na slici 6 je prikazan grafikon performansi build-ova koji uključuju testove.

3.2 Pokrivenost kôda

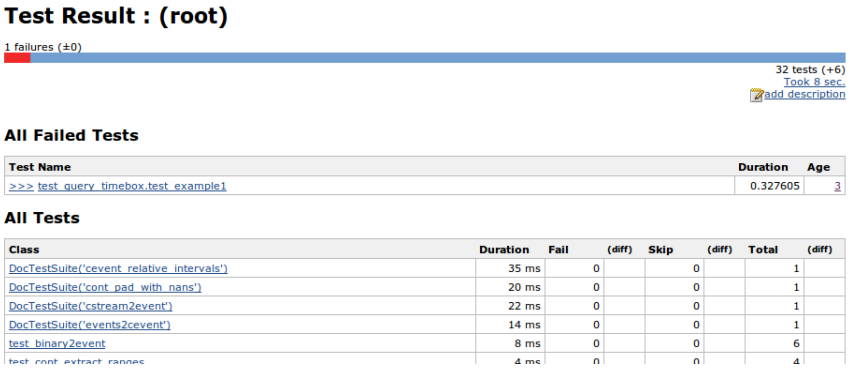
Jedna od metrika koja se ceni u testiranju softvera je *pokrivenost kôda*. Iako ona ne daje neku značajnu informaciju o kvalitetu napisanog kôda, daje informaciju koliki deo kôda je pokriven testovima jedinica. Na primer, ako naši testovi za neku funkcionalnost uvek prolaze kroz `if` granu ali ne i kroz `else` granu.

Standardni alati za testiranje pružaju opcije za pregled pokrivenosti kôda u obliku generisanih HTML stranica. Prednost integracije ove metrike u Jenkins je što je možemo pratiti iz build-a u build kroz Jenkins.

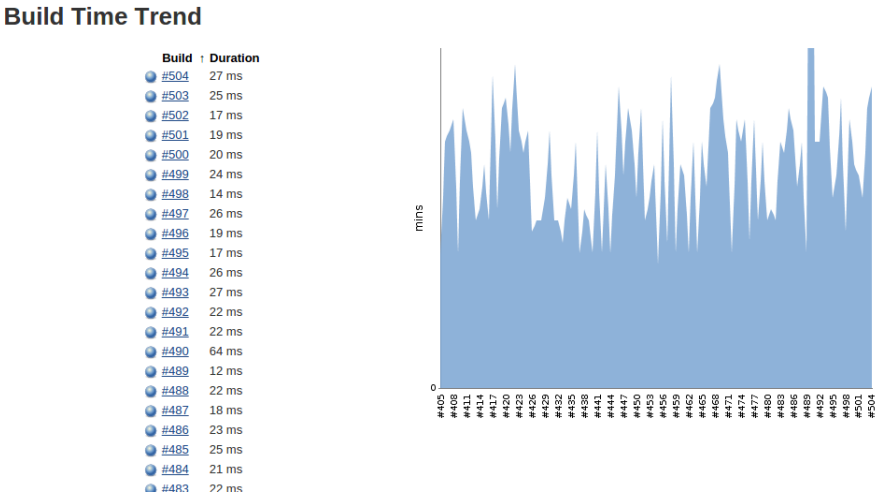
Kao i za većinu funkcionalnosti kod Jenkins sistema, potrebno je instalirati plug-in. Najpoznatiji alat za pokrivenost kôda za Javu je Cobertura



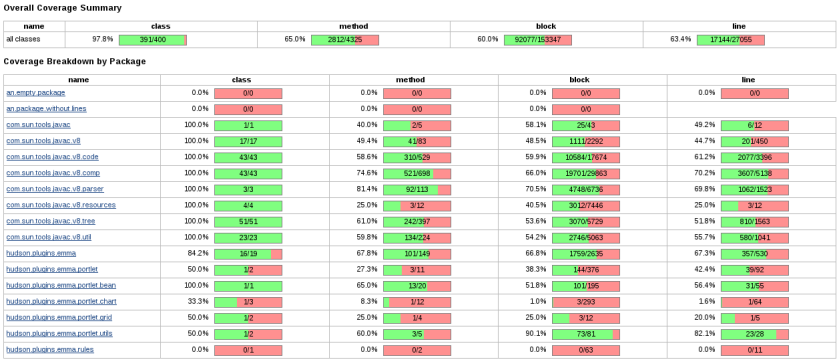
Slika 4: Početna strana projekta sa izveštajem



Slika 5: Poslednji build sa izveštajem o testovima



Slika 6: Performanse build-ova



Slika 7: Početna strana projekta sa izveštajem

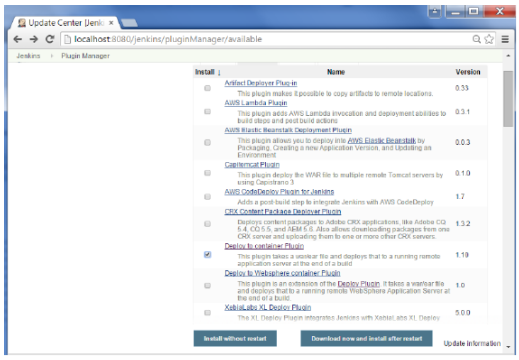
koji se uključuje u build proces. Izlaz Cobertura alata je XML fajl do kojeg je potrebno da se podesi putanja u Jenkins-u. Tada Jenkins može da iz XML fajla napravi lepe grafikone i lepše prikaže ovu metriku.

Drugi alati, kao na primer PHPUnit, u sebi već imaju ugrađene sisteme za pokrivenost kôda. S toga, nije potrebno ništa dodatno podešavati jer plugin-ovi su dovoljno pametni da to prepoznaju i integrišu u Jenkins.

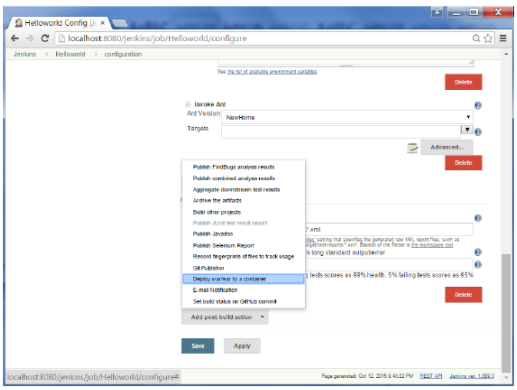
Na slici 7 je prikazano kako izgleda grafikon pokrivenosti kôda koji prikazuje Jenkins za ceo projekat. Na sledećoj slici REF možemo pogledati metriku vezanu za određeni paket, dok kad kliknemo na određenu klasu prikazaće nam se kôd te klase sa obojenom pozadinom koja nam pokazuje koji to delovi kôda nisu a koji jesu pokriveni testovima.

4 Deployment

Postoje mnogi dodaci koji se koriste da prebace build fajlove posle uspesnog kompajliranja na željenu lokaciju ili web server. Ovde ćemo to



Slika 8: Dodavanje dodatka



Slika 9: Deploy war/ear to a container dodatak

pokazati na primeru Đeploy to container Plugin".

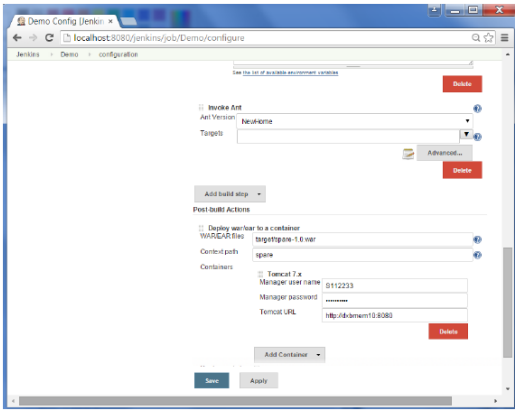
4.1 Instalacija dodatka Deploy to container Plugin

Da bi se instalirao taj dodatak, potrebno je otići na Jenkins -> Manage Plugins, pronađete željeni dodatak i instalirate ga kao što je prikazano na slici 8. Posle toga je potrebno restartovati jenkins server. Ovaj dodatak uzima fajlove i prebacuje ih automatski na server na kraju svakog builda.

4.2 Podešavanje

Potrebno je otici na Vaš Build project i kliknuti na Configure option. Izaberite opciju Đeploy war/ear to a container"kao što je prikazano na slici 9.

U polja prikazana na slici 10 unesite detalja servera na kojem želite da se fajlovi šalju i kliknite na Save dugme. Ovi koraci će obezbediti da se potrebni fajlovi nadju na serveru posle uspešnog build-a.



Slika 10: Konfigurisanje parametara

5 Jenkins vs Buildbot

Name	Platform	License	Windows builders	Java builders
Buildbot	Python	GPL(GNU General Public License)	Yes(command line)	Yes(command line)
Jenkins	Servlet Container	Creative Commons and MIT	MSBuild, NAnt	Ant, Maven, Kundo

Name	Other builders	Notification	IDE Integration	Other Integration
Buildbot	Yes(command line)	E-mail, Web, GUI, IRC(Internet relay chat)	Unknown	Unknown
Jenkins	Cmake, Gant, Gradle, Ruby, Python, Shell script and others	Android, E-mail, Google Calendar, IRC, XMPP, RSS, Twitter, Slack	Eclipse, Intelij IDEA, NetBeans	Bugzilla, Google Code, JIRA and others

6 Zaključak

Postoje mnogi benefiti korišćenja Jenkinsa, a neki od njih su:

- Momentalno otkrivanje grešaka u kodu
- Povećana produktivnost i efikasnost svih timova uključeni u razvoj, testiranje i automatizaciju procesa
- Nema integracioni korak u ciklusu razvijanja softvera
- Sistem je u svakom trenutku spreman za isporučivanje
- Čuva se redosled razvijanja projekta

- Povećan kvalitet završnog koda

Literatura

[1] John Ferguson Smart. *Jenkins: The Definitive Guide*. O'Reilly Media, 2011.

A Dodatak

Izvorni LaTeX kôd za ovaj rad se može naći na https://github.com/bokisha/mnsr.git