TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia eriala

Irina Ivanova

Versiooniuuenduse raamistik kasutades skriptimiskeelt Bash

Bakalaureusetöö (6 EAP)

Juhendajad: dotsent Helle Hein
Polina Morozova, MSc (Nortal AS)

TARTU 2016

Versiooniuuenduse raamistik kasutades skriptimiskeelt Bash

Lühikokkuvõte:

Võtmesõnad:
Bash skript, versiooni uuendus, Apache Tomcat, Java

Thesis title

Abstract:

Keywords:
Bash script, version update, Apache Tomcat, Java

Sisukord

Sissejuhatus

Käesolevas lõputöös lahendatakse rakenduslikku probleemi toimivas tarkvaraprojektis. Probleem seisneb selles, et projekti olemuse tõttu toote versiooni uuendamise protsess võtab kaua aega ja eeldab palju manuaalset sekkumist. See raiskab resursse ja suurendab inimliku vea tekkimise tõenäosust. Töö eesmärk on automatiseerida versiooniuuenduse protsessi konkreetses projektis, selleks, et lihtsustada antud protsessi, et säästa aega ja vähendada inimlike vigade arvu.

Töö koosneb viiest osast. Esimeses osas püstitatakse probleem: kirjeldatakse projekti arhitektuuri, manuaalse uuendamise protsessi, probleeme, mis vana protsess toob ja nõudeid uuele lahendusele. Peamine väljakutse seisneb selles, et toode koosneb paarkümnest moodulitest (täpne arv sõltub kliendi nõuetest), mida on vaja paigaldada 31 keskkonna. See tähendab, et selleks, et paigaldada toote uut versiooni kõikidele keskkondadele on vaja teha umbes 620 uuendust (mitte arvestades testkeskkondade vaheuuendustega). Lisaks on projektis kasutusel kaks erinevat veebiserverit: Oracle WebLogic 12 ja Apache Tomcat 8. Uuenduse protsessi automatiseerimine peab võtma arvesse kõik need väljakutsed.

Teises osas uuritakse võimalikke lahendusi: kas kasutada turul olemasolevat toodet (töös uuritakse neli toodet: Atlassian Bamboo, Chef, Jenkins ja Ansible Tower) või kirjutada ise tarkvara. Tuuakse välja iga lahenduse eeliseid ja puuduseid ning seletatakse miks langetati otsus Bash skripti kirjutamise kasuks. Valmistoodete peamised puudused on maksmus, võimetus hallata protsesse ja vajadusel parandada vigu, vajadus kulutada palju aega toode tundmiseks ja rakendamiseks.

Kolmandas osas kirjeldatakse valitud lahendust: mis on skripti tööpõhimõtte ja arhitektuur ja kuidas seda saab paigaldada ja kasutada. Skripti põhifunktsionaalsus on mooduli faili alla laadimine, vana versiooni maha võtmine, uue versiooni paigaldamine, kolleegide teavitamine toimunud versiooniuuendusest. Lisaks oluliseks funktsionaalsuseks on erinevad kontrollid, mis vähendavad tõenäosust paigaldada vale versioon ja logimise funktsionaalsus, mis võimaldab kätte saada põhjalikku infot iga uuenduse kohta.

Neljandas osas uuritakse kuidas antud lahendus mõjus projektile: võrreldakse versiooni uuendusele kulunud aega manuaalse ja automatiseeritud protsessi puhul ja lisatakse kolleegide tagasisidet uue raamistiku kasutamisele.

Viimases osas tuuakse välja lahenduse perspektiive: kuidas loodud skripti saab edasi arendada ja kuidas saab seda teistes projektides rakendada.

Töö tulemuseks on töötav, avatud lähtekoodiga, skript Bash keeles ja selle dokumentatsioon.

1 Probleemi püstitus

Versioonide uuendamine on tingimata vajalik protsess tarkvara arenduses. Projektis, mida kirjeldatakse käesoleva töö raames, see protsess mängib väga olulist rolli.

Projekti rakendus on kirjutatud Java keeles ja on kasutusel .WAR failid. Lisaks projekt rakendub modulaarsuse süsteemi, mis tähendab, et kogu rakenduse kood on jagatud moodulite kaupa ja iga mooduli kohta on olemas eraldi .WAR fail. Kokku on umbes 20 moodulit (täpne arv sõltub kliendist, sest erinevatel klientidel on erinev moodulite komplekt).

Rakendust kasutavad 10 klienti. 3-l nendes on olemas 3 keskkonda: toode, demo (kus kliendid tutvuvad uue funktsionaalsuse ja parandustega enne toote uuendust) ja test (kus projekti meeskond testib funktsionaalsust enne tarnimist kliendile). 7-l kliendil on olemas 2 keskkonda: toode ja test. See tähendab, et kokku on olemas $3 * 3 + 7 * 2 = 23$ kliendikeskkonda. Lisaks veel umbes 7 projekti test keskkonda, mida kasutatakse erinevatel faasidel erineva funktsionaalsuse testimise jaoks. Seega kokku tuleb $23 + 7 = 30$ keskkonda, mida on vaja regulaarselt uuendada (mitte arvestades lokaalsete ja virtuaalmasinatega, mida iga meeskonnaliige võib vajadusel panna püsti). See tähendab, et selleks, et paigaldada rakenduse uut versiooni igale keskkonnale on vaja teha $30 * 20 = 600$ uuendust.

Toodud numbrid näitavad, et versiooni manuaalne uuendus võtab palju aega ja tähelepanu, mille pärast oli otsustatud lihtsustada ja automatiseerida uuenduse protsessi.

Manuaalne uuenduse protsess

Ühe mooduli manuaalne uuenduse protsess koosneb järgmistest sammudest:

1. .WAR faili uue koodiga alla laadimine serveri peale.
2. Kui veebiserveriks on Oracle WebLogic, siis .WAR faili prekompileerimine (Apache Tomcat veebiserveril seda sammu ei ole).
3. Faili ümber nimetamine.
4. Faili paigaldus.
5. Vana faili eemaldamine.

6. Uue versiooni staatuse kontrollimine.

7. JIRA töö uuendamine, mis automaatselt saadab emaile kolleegidele, et versioon on uuendatud (iga mooduli ja keskkonna kohta on olemas eraldi JIRA töö).

8. Alla laaditud faili kustutamine.

Mitmete moodulite uuendamisel tuleb teha kõik need sammud iga mooduli jaoks. Aeg, mida võtavad need sammud, sõltub moodulist (kuna mõnes moodulis võib olla kaks korda rohkem koodi, kui teises, mis tähendab, et selle alla laadimine ja paigaldamine võtab kaks korda rohkem aega), aga keskmiselt see on 1.5 minutit. Kõikide moodulite uuendamine võib võtta tund aega, kuna lisandub veel aeg, mida inimene raisab õige versiooni numbri leidmiseks, .WAR faili aadressi kopeerimiseks, õige JIRA töö leidmiseks ja avamiseks jne. See tähendab, et inimlike vigade tekkimise tõeäosus on suur (nt alla laadida vale versiooni).

Nõuded lahendusele

- Uuenduse protsess peab võtma vähem aega.
- Uuenduse protsess peab nõudma nii vähe käsitsi tegevusi, kui on võimalik.
- Lahendus peab võimaldama uuendada nii üht moodulit, kuid ka mitu moodulit korraga.
- Projekt kasutab kaks erinevat veebiserverit: Apache Tomcat 8 ja Oracle WebLogic 12. Lahendus peab töötama sama moodi kõikidel serveritel, sõltumata sellest, mis tarkvara seal on paigaldatud.
- Mõned meeskonnaliikmed soovivad saada teavitusi toimunud uuenduste kohta, mis tähendab, et lahendusel peab olema teavituste süsteem.
- Peab olema võimalus vaadata kes ja millal uuendas mingit moodulit mingis keskkonnas, mis tähendab, et peab olema logimise süsteem.
- Uuendustega tegelevad testijad, seega protsess ei tohi olla liiga tehniline.

- Uuendustega võivad tegeleda mitu inimest samal ajal, mis tähendab, et lahendusel peab olema lukustamise süsteem, et vältida paralleelselt uuendust samal serveril.
- Toodang keskkonnad kasutavad 2 serverit, seega lahendus peab oskama uuendada versiooni automaatselt kahel serveril.
- Lahendus peab olema täielikult projekti kontrolli all, selleks, et selle haldamine, seadistamine ja parandamine oleks võimalik teostada igal ajal.
- Lahenduse loomine ei tohi võtta palju aega, sest projektil puuduvad selleks inimresurssid.

2 Võimalikud lahendused

Uuenduse protsesside automatiseerimiseks ja haldamiseks on olemas palju valmistoodet. Kõige populaarsemad nendest on Atlassian Bamboo, Chef, Jenkins, Ansible Tower ja Puppet.

Kõikide valmistoodete rakendamisel alati on olemas oma eelised ja puudused.

Eelised:

- Ei ole vaja kulutada aega süsteemi kirjutamisele, mis tegelikult on juba leiutatud.
- Kuna rakendus on juba turul ja on populaarne, siis on suur tõenäosus, et see on kvaliteetne ja vastab kasutajate ootustele.
- On olemas kogukond tarkvara kasutajatest, kes vajadusel saavad aidata ja jagada enda kogemust.

Puudused:

- Suurem osa valmistoodetest on tasulised (Bamboo ja Jenkins on erandiks) ja hind alati sõltub serverite, keskkondade ja moodulite arvust, mis vaadatud projekti korral on suur.
- Valmistoode õppimine, paigaldamine ja seadistamine võtab palju aega, eriti kui see pakkub lai valik funktsionaalsust.
- Projekt väga sõltub tootjast - vigade olemas olul ei saa olla kindel, et neid parandatakse lähimal ajal või parandatakse üldse; probleemide tekitamisel ei saa olla kindel, et tootja kasutajate tugi saab aidata.
- Projekt peab usaldama tootjat - usaldama, et valmistoode on turvaline ja stabiilne.

Need on ühtlased eelised ja puudused kõikide valmistoodete kohta. Iga tootel on olemas aga veel eriomadused.

Atlassian Bamboo [6]

Eelised:

- Tasuta, sest projekt juba kasutab teised Atlassian tooted.

- Kuna projektis kasutatakse Atlassian toodeid (JIRA, Confluence, Fisheye), siis Bamboo väga hästi integreerub nendega. On võimalik mudagvalt teha seoseid rakenduse koodi, muudatuste kirjelduse ja uuenduse vahel.
- Antud toodet kasutavad teised projektid firmas, mis annab võimalust kaasata inimesi, kellel on olemas teadmised ja kogemus Bamboo seadistamise kohta.
- Mugav kasutamine veebilehitseja kaudu.

Puudused:

- Seadistamine toimub moodulite kaupa (igal moodulil on oma Bamboo projekt) ja ühe mooduli seadistamine võtab palju aega. Firma teises projektis see aeg oli 1.5 kuud, mis antud projekti jaoks tähendaks 20 * 1.5 = 30 kuud.
- Üks uuendus võib võtta palju aega, sest Bamboo tööpõhimõte eeldab kolmanda serveri kasutamist uuenduse tööplaani teostamiseks ja selliste serverite arv on piiratud. See tähendab, et kui korraga soovitakse uuendada mitu keskkonda, siis võib tekkida ootusjärjekord.

Chef [7]

Eelised:

-

Puudused:

-

Jenkins [8]

Eelised:

- Tasuta.
- Avalik kood, mis tähendab, et projekt ei sõltu tootjast.
- Firmas on olemas spetsialistid, kes on kasutanud Jenkins-t ja saavad jagada teadmisi ja kogemust.

Puudused:

-

Ansible Tower [9]

Eelised:

-

Puudused:

-

Puppet [10]

Eelised:

-

Puudused:

-

Bash skriptimine [1]

3 Lahendus

3.1 Skripti arhitektuur ja tööpõhimõte

3.2 Skripti paigaldus

3.3 Skripti kasutus

3.4 Dokumentatsioon

4 Mõju projektile

4.1 Statistika

Module Name	Script Time (sec)	Manual Time (sec)
admin	27.245	80.45
treatment	61.101	112.71
reception	38.538	100.19

[11]

4.2 Kolleegide tagasiside

Martin Rakver, Hosting Services and Application Manager

I have not been using the script very much, because I rarely update modules nowadays. But what I can say is that it is definately in the right place – automating activities that testers would need do to on daily basis each time when deploying a new module. Come to think about it, there are actually quite many activites related to new module deployment – would be interesting to compare the time it takes to deploy a module manually vs using script + calculate about how much time we as a team are saving daily,montly,yearly (and get to do more important things with that time). I also like that scalability has been considered when writing the script + many optional features have been implemented which can be used if one desires to do so.

Kalle Jagula, QA Specialist

I personally think that version-update script has significantly improved the speed of version update. Especially with the settings for bulk-update, which improves the speed of modules' version update and manageability of test/demo environments.

Tanel Käär, System Architect

Version updater script has standardized the way our environments are updated, reducing learning curve for new people performing the task. Downside is that people are unaware, what happens in the background and can't handle simple errors in the process anymore.

Klaus-Eduard Runnel, Senior Programmer

I have mostly used the scripts to perform quick module and dependency updates in specific test environments when deploying quick fixes or debugging problems in the modules I'm responsible for. Deployment of artifacts is secondary task in such scenarios. The scripts allowed me to conveniently perform the updates while letting me concentrate on solving the primary problems and not leaving the environment containing all necessary information.

In some cases it was necessary to deploy custom-built experimental (unversioned) war-files. In these cases the scripts could not be used. They would have been useful though if such experiments would have been commited to feature branches and published as feature branch artifacts. (As of today, we are better prepared for such circumstances.)

The scripts were also useful to deploy predefined sets of modules to temporary test environments living on reusable virtual machine images. It is important to note that a module update triggers launching a set of associated sql scripts to bring a database schema up to date. In that way the version update scripts let us automate most of a process of setting up updated environment from virtual machine images.

It would be helpful if the scripts were adapted to perform changes on remote machines and to perform changes on multiple environments simultaneously.

Helina Ziugand, QA Specialist

Uuendamise skriptid on nii projektile kui ka minu igapäevasele tööle väga positiivselt mõjunud. Kui algselt oli uuendamise jaoks vaja mitu sammu käsitsi teha, siis nüüd käib see automaatselt ja kiiresti. Võiks öelda, et üsna tüütu oli alguses kogu see uuendamise protsess - pidevalt pidi jälgima, kas vaja käsureale kirjutada järgmist sammu, hiljem JIRAs vana versiooni numbrit muuta ja Weblogicust vana versioon kustutada. Praegu on väga mugav ühe käsklusega uuendamise skript tööle panna, mis korraga kõik vajalikud sammud ära teeb ja samal ajal kui skript jookseb ise teiste tegevustega jätkata. Väga selgelt erinevate värvidega on välja tootud error, warningu ja success teated, mis on lihtsasti märgatavad ja hästi loetavad.

Kui vaja on rohkem kui ühte moodulit uuendada, siis väga hea lahendus on selleks mitme moodulise uuendamise skript. Kui varem oli vaja keskkonda

uue tsükli peale uuendada, siis võttis see ebamugavalt kaua aega ja vabatahtlikult kõige parema meelega seda ette ei tahtnud võtta. Nüüd on vaja ainult uuendatavad versioonid tekstifaili kirja panna ja ühe käsklusega skript käima tõmmata.

Samuti tooks välja, et uuendamise skriptid on väga hästi dokumenteeritud ja põhjalikult kirja pandud kuidas toimivad. Mul on väga hea meel, et versioonide uuendamine on skriptide abil nii lihtsaks, mugavaks ja meeldivaks tegevuseks saanud.

5 Lahenduse perspektiivid

Kokkuvõte

Kokkuvõttes tuuakse selgelt välja töö põhilised saavutused. Ei tohi sisse tuua uusi väiteid, põhjendusi ja analüüse, mida töö põhiosas ei ole käsitletud. Mõistlik on viidata eesmärgile ja sõnastada kokkuvõte nii, et on näha eesmärgi saavutamine ning ka autori panus.

Kokkuvõttes võib lühidalt esile tuua töö edasiarendamise võimalikke teid ja perspektiive.

Kokkuvõtte tekstis on soovitav kasutada lihtmineviku umbisikulist tegumoodi, näiteks "Töös toodi välja ..., kirjeldati ..., leiti lahendus ... ", aga ka olevik on vastuvõetav.

Kasutatud materjalid

[1] https://www.gnu.org/software/bash/manual/bashref.html

[2] https://tomcat.apache.org/tomcat-8.0-doc/manager-howto.html

[3] https://help.ubuntu.com/community/CronHowto

[4] https://www.mercurial-scm.org/guide

[5] https://wiki.centos.org/FrontPage

[6] https://www.atlassian.com/software/bamboo

[7] https://www.chef.io

[8] https://jenkins-ci.org

[9] https://www.ansible.com

[10] https://puppet.com

[11] http://pubs.opengroup.org/onlinepubs/9699919799/utilities/time.html

Lisad

5.1 Terminid

Rakendus –
Server –
Keskkond –
Moodul –
Versioon –
Modulaarne süsteem –
Veebiserver –

- Skripti manuaal: `https://iriiiina.gitbooks.io/version-updater-manual/content/`
- Skripti kood: `https://github.com/iriiiina/version-updater`
- BPMN diagramm ühe mooduli uuenduse kohta Tomcat-i peal (update-version-tomcat.sh): `https://github.com/iriiiina/bachelors-thesis/blob/master/thesis/pictures/BPMN-diagram-one-module-tomcat.png`