TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Margus Koval 001363IASB

VEEBIRAKENDUSTE KOOSTAMINE JAVA KESKKONNAS AVALIKU SEKTORI ETTEVÕTTE NÄITEL

Bakalaureusetöö

Juhendaja: Vladimir Viies
Dotsent

Tallinn 2020

Bakalaureusetöö ülesanne

Üliõpilane: Margus Koval, 001363IASB
(nimi, üliõpilase kood)

Töö teema: Veebirakenduste koostamine Java keskkonnas avaliku sektori ettevõtte näitel

Töö teema inglise keeles: Compiling Java web applications in a Public Sector Institution

Teema päritolu: Avaliku sektori asutus, Riigi Kinnisvararegister

Juhendaja: Vladimir Viies, PhD, Arvutisüsteemide instituut
(nimi, teaduslik kraad, töökoht)

Kaasjuhendaja: -

Konsultandid: Martin Lindjärv, Kalle Pärtlas

Lähtetingimused: Pidevintegratsiooni töövahend, lähtekoodihoidla, andme-ja rakendusplatvormi kiht virtuaalserveritel, rakenduste lähtekood

Lahendatavad küsimused vastavalt õpiväljunditele:

A) Süsteemsed aspektid: Tarneahela optimeerimise metoodika

B) Tarkvaralised aspektid: Tarneahela protsesside automatiseerimine Java keskkonnas

C) Riistvaralised aspektid: Kaudselt kaasatud, platvormi aluskihina

Eritingimused: puuduvad

Nõuded vormistamisele: vastavalt TTÜ Infotehnoloogia teaduskonna nõuetele

Bakalaureusetöö esitamise tähtajad: **11.01.2021** **19.01.2021**
(elektroonselt: jane.rang@ttu.ee) *(köidetult: ICT-525)*

Ülesande vastu võtnud:
(lõpetaja allkiri, kuupäev)

Kinnitanud:
(nimi, allkiri, kuupäev)

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Margus Koval

Annotatsioon

Antud bakalaureusetöö eesmärk on optimeerida avaliku sektori asutuse pideva integratsiooni ahelas toimuvaid protsesse, vähendades seeläbi käsitsi tehtavate muudatuste mahtu ning tõstes rakenduste tarnimise võimekust.

Eesmärkide saavutamiseks pakub autor lahendusi kuidas paremini automatiseerida Java brauserrakenduste kompileerimist, testimist, seadistamist ja paigaldamist. Lisaks uurib autor võimalusi kuidas suurendada antud protsesside läbipaistvust ning protsessidest saadavat teavet ära kasutada seiresüsteemides.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 18 leheküljel, 3 peatükki, 6 joonist, 1 tabelit.

Abstract

Compiling Java web applications in a Public Sector Institution

In this day and age when it comes to web-based based information systems the common trend is to move from a monolithic to more of a service-based architecture (known as micro services). A service-based architecture should offer higher availability while making it easier to implement changes without experiencing downtime. According to State of Devops 2019 - a report done by Devops Research and Assessment (DORA) successful companies or organizations are the ones that are able to deliver and implement changes faster while reducing the amount of manual labor and maintenance windows [1]. The aim of this thesis is to find ways to optimize and automate the processes of building, configuring, testing, deploying and monitoring Java web applications in a Public Sector Institution.

The Institution is managing around four hundred Java web applications and the steady flow of new applications has created a situation where it is impossible to rely on manual work and custom solutions – especially when a task could be done automatically. All the bottlenecks of processes have created a situation where the timelines of projects and 3rd party clients are affected due to lack of feedback and the inability to implement changes within an agreed time frame.

Making changes to processes will also trigger changes in work culture - the nature of responsibilities and freedom between developers and system administrators is becoming more fused. In this thesis the author will discover adequate wins for both sides when optimizing the continuous integration pipeline.

The thesis is in Estonian and contains 18 pages of text, 3 chapters, 6 figures, 1 table.

Lühendite ja mõistete sõnastik

SSH	*Secure Shell.* Võrguprotokoll - võimaldab turvalist krüpteeritud pöördumist operatsioonisüsteemi poole. [14]
HTTP	*Hypertext Transfer Protocol.* Teksti edastuse protokoll mida kasutab WWW defineerimaks kuidas sõnumid on edastatud ja vormindatud ning kuidas veebiserverid ja veebilehitsejad peavad reageerima erinevatele käsklustele [15]
Pidevintegratsioon	*Continuous Integration.* Automatiseerimise praktika mis võimaldab integreerida erinevate osapoolte lähtekoodi üheks tarkvaraprojektiks [16]
Ülalhoid	Struktuuriüksus mis on spetsialiseerunud infrastruktuuri eri kihtide, komponentide ning neid kasutavate teenuste administreerimisele.
Artifakt	*Artifact.* Pakendatud tarkvarapakett mida saab kasutada testimiseks, paigaldamiseks või edastamiseks. [18]

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Tänapäeva infosüsteemid on liikumas monoliitse arhitektuuri pealt suunale mis koosneb üksteisest sõltumatutest mikroteenustest. See tagab parema käideldavuse ja loob võimaluse muudatuste tegemiseks kiiremini ja süsteeme halvamata. Vastavalt Puppetlabs State of Devops 2017 aruandele [2] eristuvad edukad ettevõtted keskpärastest just võime poolest tarnida ja paigaldada teenuseid kuni 30 korda kiiremini, katkestusevabalt ja minimaalse käsitööga. Teenusepõhisele arhitektuurile liikumine pole aga universaalne valem probleemide lahendamiseks – seda peavad toetama muudatustega kiirelt kohanev infrastruktuur, spetsialistid kui ka kasutatavad töövahendid. Antud töös analüüsib autor just võimalusi kuidas optimeerida infosüsteemide kompileerimist, seadistamist, testimist ja paigaldamist avaliku sektori asutuses.

Asutuse hallata on sadu erinevaid veebirakendusi ja infosüsteeme kuid seni pole tähelepanu suuremat pööratud protsessidele endile – tekkinud on olukord kus lihtne muudatus või veaparandus ootab tundide kaupa inimressursi taga, samas kui ülesande enda tegemist peaks olema võimalik automaatikaga (ning ilma inimliku eksimuse faktorita) lahendada. See omakorda hakkab mõjutama allasutuste tööd ning projektide ajaraamides püsimist - tekib olukord kus arendaja ei saa enda tarnete kohta mõistliku ajaraami sees tagasisidet sest allüksusel pole olnud võimalik tarnega kaasnenud muudatusi valideerida.

Muudatused protsessides omakorda tingivad muudatusi asutuse enda töökultuuris – monoliitsete rakenduste haldus on seadnud väga ranged piirid vastutusele ning võimalustele teha jooksvalt rakenduses väikseid muudatusi. Implementeerides aga asutuses Devops kultuuri põhimõtteid ning suurendada automaatikat vastutuse piir hägustub – sellega seoses toob autor töös välja võimalikud võidud kõigile osapooltele kasutades Lean metoodikate põhimõtteid.

Lõputöö on jaotatud neljaks osaks – autor alustab Devops kultuuri ja Lean metoodikate tutvustuse ning analüüsiga, sellele järgneb asutuse protsesside analüüs ning leitud

vajadustest lähtuvalt eesmärkide defineerimine. Peale seda kirjeldab autor protsesside optimeerimise lahenduskäike ja viimaseks annab ülevaate saadud tulemustest.

1 Rakendusmooduli koostamine

Enne asutuse enda töövahendite ja protsesside analüüsi uurib autor agiilse tarkvaraarenduse, Devops kultuuri ja Lean praktikate aluseid. Vastavat analüüsi kasutades leiab autor parimad võimalikud tavad mida asutuse sees realiseerida ilma et sellega kaasneks struktuurimuudatused või rollide ja kohustuste muudatused mis peaks kajastuma spetsialiste ametijuhendites.

1.1 Devops kultuur ja põhimõtted

Mõistest „Devops“ on saanud viimastel aastatel moesõna mida iga kasutaja mõistab enda vaatenurgast ning mida on defineeritud tuhandeid erinevaid viise. Üks suuremaid väärarvamusi mis mõistega kaasneb on autori arvates põhimõte et Devopsi kasutamine hõlmab Devops inseneride (spetsialistide kes on kompetentsed nii arenduses kui halduses) värbamist ja/või asutuse struktuuri muutmist vastavalt. Autor näeb Devopsi ennekõike kui kultuuri mis lähtub järgmistest põhimõtetest:

- Agiilse tarkvara tarnimise protsessiga peab kaasnema defektide vähenemine ja ressursside raiskamine [3].
- Loodud on projektipõhine töökeskkond kuhu on kaasatud kõik projektis osalejad – arendajad, ülalhoid, projektijuhid, kvaliteedikontroll jne. [3]
- Eesmärk on muuta IT lihtsamaks, kiiremaks ja odavamaks, samal ajal luues lisaväärtust kliendi või kasutaja jaoks [7]

Agiilsust Devops kultuuri vaatenurgast mõistetakse kui organisatsiooni võimet kohaneda pidevate muutuste ja klientide soovidega [4] – omadused mis on IT sektoris kriitilised. Puppetlabs poolt läbi viidud „State of Devops 2017“ uuring tuvastas et kõrge jõudlusega IT organisatsioonid teevad paigaldusi 30 korda tihedamini, vajavad 200 korda vähem aega muudatuste ette valmistamiseks, omavad 60 korda vähem katkestusi ning on võimelised katkestuste korral teenuse töö taastama 16 korda kiiremini kui madala jõudlusega organisatsioonid [1]. Need omadused toovad omakorda kaasa positiivse efekti tarkvara arenduse jaoks kes saavad lisandunud vaba aega kasutada selleks et produtseerida uut koodi ning lisada rakendustele uusi omadusi [5].

Olemuselt soovib iga rakenduse tellija et tarneprotsessi mis toimuks kiirelt ja pidevalt. Traditsioonilises töökeskkonnas põrkub see soov aga spetsialiste erinevate rollide taha – tarkvaraarendus soovib luua ja tarnida tarkvara mida klient tellinud on, ülalhoid aga vastutab platvormi ees millel rakendused jooksevad. Just ülalhoiu ülesanne on tagada taristu ja sellel jooksvate rakenduste stabiilsus ja käideldavus. Erinevad eesmärgid ja nägemused loovad aga meeskondade vahel näilise seina mille tagant teist poolt ei nähta ning psühholoogilised ja protseduurilised barjäärid takistavad efektiivset kommunikatsiooni erinevates projekti faasides. Tulemuseks on mahajäämus ja aeglus võrdluses konkureerivate organisatsioonidega. Devopsi juurutamise põhimõtteid on erinevaid kuid antud töös keskendub autor neist kahele – pidevale arengule (Continuous Improvement) ning protsesside automatiseerimisele.

1.1.1 Pidev areng

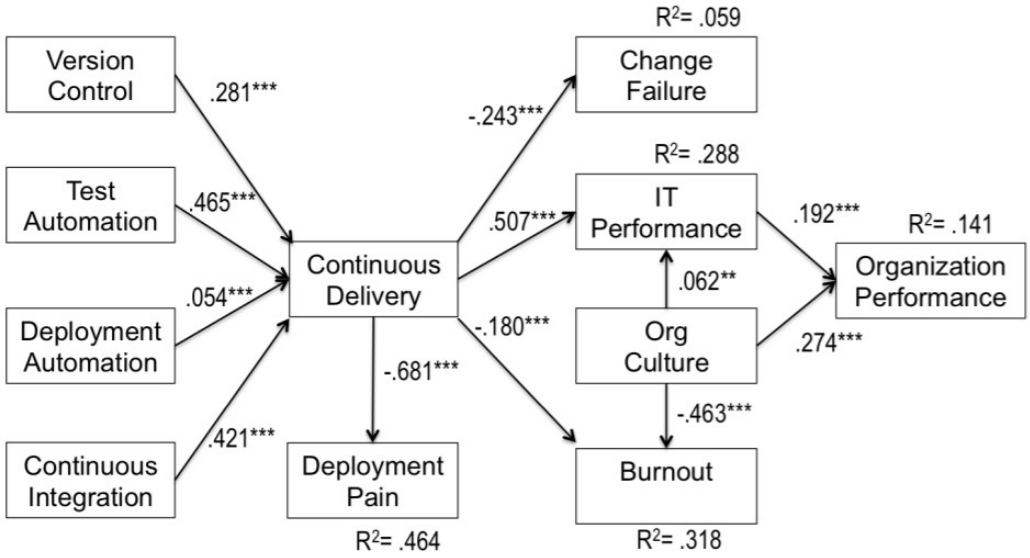
Pidevaks arenguks nimetatakse põhimõtet mille käigus leitakse viise kuidas teha tööd kiiremini ning vähendada prügi – antud töö käigus kas siis defekte, kasutamata oskusi, või ooteaega. Selleks peab organisatsioon soodustama eksperimenteerimist ja võimaldama õppida meeskondadel kaasnevatest tagasilöökidest. Arengu hindamiseks tuleb implementeerida vastavad mõõdikud jõudluse, protsesside, innovatsiooni, kultuuri ja kasutajatoe tarbeks. Selle kõige eelduseks on organisatsiooni kultuur – ilma selleta on pidev areng olemuselt raha, aja ja motivatsiooni raiskamine [6]

1.1.2 Protsesside automatiseerimine

Iga rakenduse või teenuse taga on mingisugune protsess – kas manuaalne või automaatne. Liikudes manuaalselt töölt automaatsele suureneb kvaliteet, efektiivsus ja protsesside voolavus ning pidevus. Levinumad põhimõtted automatiseerimiseks on:

- Pideva arengu kultuuri juurutamine aitab kaasa automatiseerimisele ja raiskamise vähendamisele
- Tarneahela optimeerimine võimaldab teha muudatusi mitu korda päevas ilma inimressurssi raiskamata
- Pilveteenuste kasutamine võimaldaks vähendada serveriparkide ülalhoidu
- Konteineripõhine infrastruktuur võimaldab taristu kirjeldamist lähtekoodina

Tarnehala optimeerimise ja automatiseerimise mõju nii IT enda jõudlusele kui ka organisatsioonile on analüüsinud Nicole Forgensen ja Jez Humble uurimustöös [10]. Töös loodud mudeli järgi toetavad nii organisatsiooni kultuur kui automatiseerimine otseselt ettevõtte jõudlust.



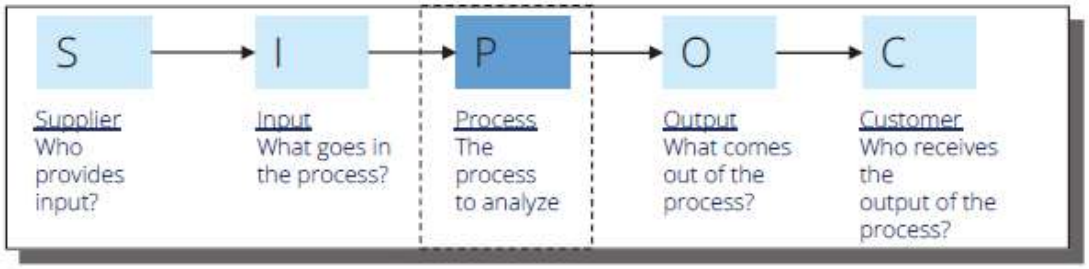
Joonis 1. Pideva tarne protsessi mõjutavad tegurid [10]

1.2 Lean põhimõtted

Lean on olemuselt viis kuidas mõelda ja käituda eesmärgiga tõsta kliendi rahulolu ning strateegilist ja finantsilist kasu. Kaks Lean metoodika tööriista mida autor töös kasutab on väärtuste voo kaardistus (Value Stream Mapping) ning SIPOC mudel.

1.2.1 SIPOC

SIPOC (Suppliers, Inputs, Process, Outputs, Customers) on tööriist protsesside ulatuse määramiseks ning aitab meeskondadel mõista protsessi alguse ja lõpu olemust. Lisaks on läbi tööriista kergem defineerida protsesside sisendeid, väljundeid ja nende päritolu. Protsessi enda sammude mõistmiseks ja mõõtmiseks tehakse SIPOC kolmandas faasis väärtuste voo hindamine (Value Stream Mapping)

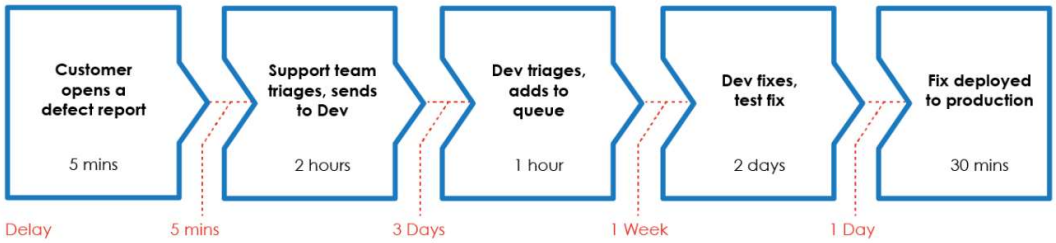


Joonis 2. SIPOC mudel [20]

1.2.2 Value Stream Mapping

Value Stream Mapping ehk väärtusvoogude kaardistus on tehnika mille läbi analüüsida, disainida ja juhtida materjalide ja informatsiooni voogusi mida on vaja et toode tellijani jõuaks[12].

Olemuselt on Value Stream Mapping viis kuidas tuvastada protsessides ressursside raiskamist ning visualiseerida tegevuste sammud mis lõpptulemuse saamiseks läbitakse. Value Stream Mapping peab olema piisavalt detailne et protsesside samme analüüsida. Analüüsi käigus peab iga protsessi osa saama ajalise väärtuse, sealhulgas ka sammud antud osade vahel – sel viisil saab hinnata tsükli efektiivsust ning omada mõõdikut selle hindamiseks.



Joonis 3. Väärtusvoogude kaardistus [11]

Väärtusvoogude kaardistuseks kasutatakse erinevaid mõõdikuid [20]:

- Täitmise aeg – aeg mis kulub kogu väärtusvoo läbimiseks

- Töötsükli aeg – aeg mille käigus tehakse manuaalset tööd
- Ooteaeg – aeg mille käigus protsess ootab kas inim-või masinressursi järel
- Masinaeg – aeg mille käigus toimub automatiseeritud töö
- Vahetuse aeg – aeg mis kulub masina ümber seadistamiseks

Väärtusvoo efektiivsuse mõõtmiseks jagatakse töötsükli ja masinaja summa töö täitmise ajaga, tulemust mõõdetakse protsentuaalselt.

1.3 Avaliku sektori asutus, protsessid

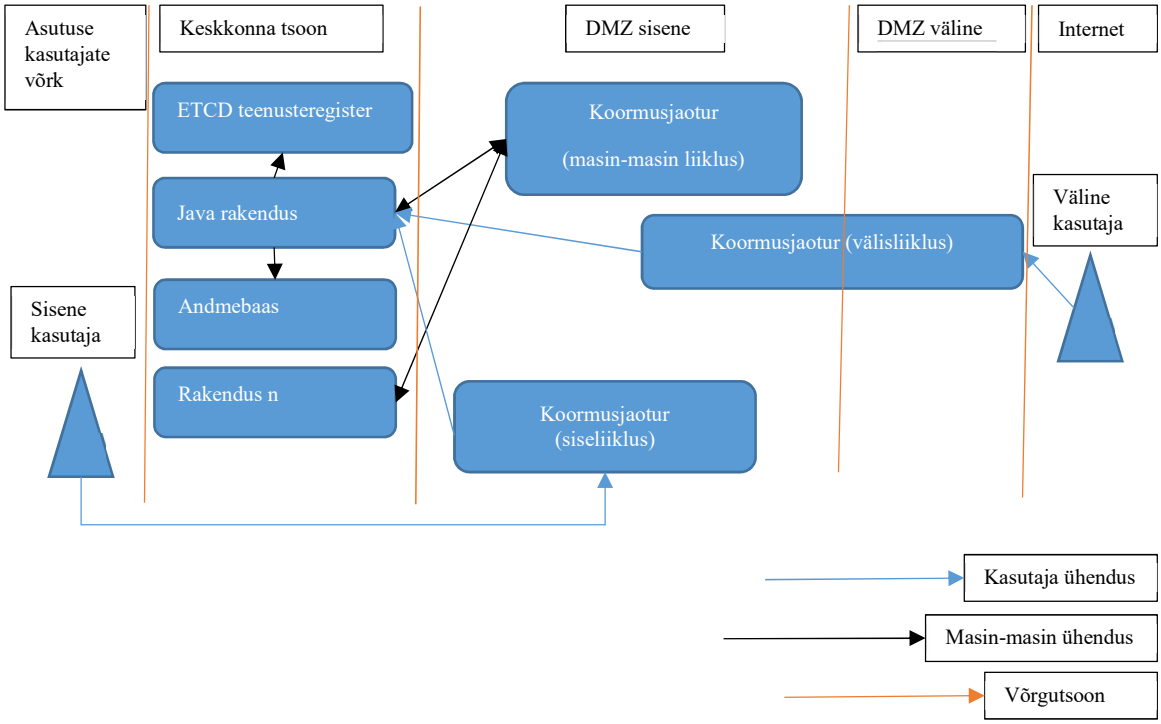
Antud töös analüüsib autor avaliku sektori asutuse protsesse mis puudutavad Java infosüsteemide ja brauserrakenduste koostamist olemasoleva lähtekoodi ja teekide baasil. Autor jätab kirjeldusest kõrvale lähtekoodi ning teekide loomise ja üles laadimise vastavalt lähtekoodihoidlasse ning teegihoidlasse.

Asutusel on kasutusel versioonihaldustarkvara Git kuhu sisearendajad ning välispartnerid lähtekoodi laevad. Vastavalt arenduslepingule kuulub lähtekood asutusele ning versioonihaldustarkvara kaudu on võimalik omada ülevaadet tarnetes tehtud muudatustest, teha vajadusel koodiauditit ning omada võimalust teha jooksvalt rakendustes parandusi kord kui välispartneriga on vastav leping infosüsteemi arenduseks lõppenud.

Pidevintegratsioonivahendiks on asutusel olemas Atlassian Bamboo mille kaudu süsteemiadministraatorid saavad lähtekoodi versioonihaldusest alla laadida, teostada kompileerimist ning paigaldada ehitatud artifakt rakendusserveritele - selle eelduseks on vastavatele rakendustele ehitus-ja paigaldusplaanide loomine. Rakendusserverikihi platvormideks on Oracle Weblogic või Apache Tomcat, veebiserveriks Apache HTTP ning andmeplatvormiks Oracle Database või PostgreSQL. Vastavalt rakenduskihile on alusplatvormiks Centos 7 või Redhat Linux. Antud serverid jooksevad Xen või Vmware virtuaalkihil mis omakorda jooksevad üldiselt kasutavate serverite riistvara peal, Intel platvormil.

Joonis 4. Asutuse infrastruktuuri kihid

Asutuse poolt hallatav veebirakenduste arv küündib viiesajani ning sõltuvalt teenuse tellijast või tarbijast on iga rakendus paigaldatud kahte kuni viite erinevasse keskkonda. Keskkonnad töötavad üksteisest sõltumatult (välja arvatud teatud keskkondade sõltuvus ühisest riistvarakihist) ning teineteisele ligipääse omamata. Iga allasutuse iga keskkond omab eraldiseisvat Apache HTTP serverit mis toimib ka koormusjaoturina. Jaoturis on defineeritud erinevate võrgureeglitega virtuaalserverid sõltuvalt sellest kas rakenduses toimub masin-masin liiklus serverite vahel, kas rakendus on asutusesiseseks kasutamiseks või on tegu rakenduse avaliku osaga. Kogu suhtlus rakenduste endi vahel käib üle masin-masin liikluse koormusjaoturi samas kui väliskasutajate ja sisekasutajate jaoks eraldiseisvad koormusjaoturid/HTTP serverid. Serverid ja koormusjaoturid asetsevad eraldiseisvates võrgutsoonides. Liikluse ja võrgutsoonide kirjeldamiseks on autor lisanud tööle joonise 5.



Joonis 5. Võrgutsoonid ning kasutajate ja masin-masin liiklus infosüsteemi näitel

Koormusjaoturile rakenduste asukohtade defineerimise lihtsustamiseks on kasutusel arenduse poolt loodud vastav teek ning register. Rakendused mis seda kasutavad suudavad iseseisvalt end koormusjaoturis registreerida, tulemuseks on vähem käsitööd, eksimisruumi ning ebastandardseid lahendusi. Vastav register töötab kasutades etcd vabavaralist võtmehoidlat [8] mille sisalduvat infot pärib confd ning produtseerib sellest koormusjaoturi seadistuse. Läbi etcd ja confd on võimalik ka rohkema info kui rakenduste asukoha produtseerimine – saadavat infot saab ära kasutada koormusjaoturis defineeritud rakendusteste kaardistuseks [9].

Rakenduste seireks kasutab asutus vabavaralist Zabbix montooringuvahendit. Infosüsteemide platvormide monitooring on suurest automatiseeritud läbi konfiguratsioonihalduse, kuid infosüsteemide enda monitoorimine on hetkel lahendatud läbi seirelehtede käsitsi lisamise kaudu Zabbixisse.

Asutuse poolt hallatavate rakenduste lähtekoode hoitakse Gitlab koodihoidlas. Vastavalt kokkuleppele kuulub kogu tarnitav kood asutusele ning kõik tarnitavad rakendused tuleb ehitada kokku asutuse enda poolt. Kompileerimiseks vajalikud sõltuvused peavad arendajad laadima teegirepositooriumisse Nexus. Teegirepositooriumi läbi saab

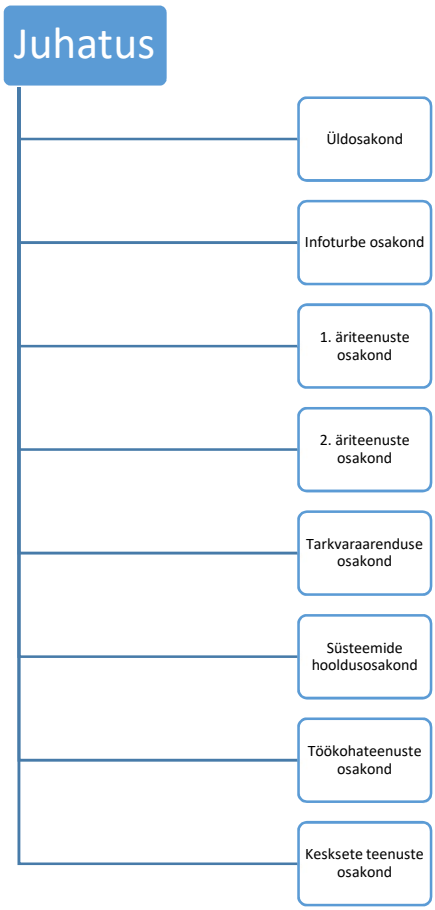
pidevintegratsioon ühenduse ka Maven keskse repositooriumiga internetis, kuid sellega ka ühendus välismaailmaga piirdub.

Kompileerimise ja paigalduse automaatika asub eraldi vastavalt ehitus –ja paigaldusserveritel. Ühendust nende vahel haldab pidevintegratsiooni moodul – eesmärgiks on turvalisuse ja käideldavuse eesmärgil protsessid lahutada ning ebavajalikud ühendused sõltuvalt protsessist blokeerida. Erinevalt paigaldusserveritest ei oma ehitusserver ligipääsu rakendusserveritele ja andmebaaside, samas kui paigaldusserver ei oma ühendust väljaspool sisevõrku olevaid süsteeme.

2 Brauserrakenduse koostamine avaliku sektori asutuses

2.1 Asutuse kirjeldus

Asutuse näol on tegu avaliku sektori ettevõttega kes pakub erinevaid infotehnoloogia arenduse ja haldusega seotuid teenuseid teistele riigiasutustele. Asutuse klientideks on riiklikud ettevõtted kelle IT on konsolideeritud antud asutuse loomise käigus. Asutuses töötab 180 täiskohaga töötajat, infosüsteemide koostamise ja seirega on seotud erinevad süsteemiadminstraatorite, rakendusadministraatorite ning tehnilise toe meeskonnad, neid omakorda toetavad koos rakendusadministraatoritega töötavad projektijuhid. Lisaks töötavad asutuses arendajad ning kesksete teenuste ja infoturbe osakondade spetsialistid. Äriteenuste osakonnad kus töötavad projektijuhid ning rakendusadministraatorid on jagatud vastavalt klientidele kaheks eraldi toimivaks struktuuriüksuseks.



Joonis 6. Asutuse struktuur

Klientide puhul on täheldada soovi suurendada pakutavate teenuste arvu kui ka kaasajastada olemasolevaid infosüsteeme, millest suur osa on ehitatud monoliitsest

arhitektuurist lähtuvalt. Mõlemad toovad endaga kaasa rakenduste arvu kasvu ja koos sellega soovi et uuenduste tarnimine toimuks sujuvamalt ja läbipaistvamalt kui seni.

2.2 Hetke protsessi kirjeldus

Rakenduste ülalhoiu, seadistamise ja paigaldamisega seotud operatsioonid on jagatud kahe osakonna meeskondade vahel. Ülalhoiu meeskond vastutab rakendusserverite kihi eest ning teostab rakenduste kompileerimist, seadistamist ja paigaldamist läbi pidevintegratsioonivahendi. Paigaldatud ja käivitatud rakenduste jooksva haldusega tegeleb teises osakonnas töötav rakendusadministraatorite meeskond.

Infosüsteemide arendajad tarnivad lähtekoodi ja teeke iseseisvalt vastavalt arendusnõutele lähtekoodi –ja teegihoidlatesse ning teavitavad projektijuhti tehtud tarnest läbi projektihaldustarkvara. Antud teavitus peab sisaldama tarnes tehtud muudatusi ning infot ülalhoiu meeskonnale kas rakenduse seadistamine, ehitamine või paigaldamine on võrreldes eelmise tarnega muutunud. On kriitiline et tarne kirjeldusse jõuaks ainult vajaminev, vastasel juhul peab ülalhoid raiskama aega ning hakkama dokumentatsiooni või paigaldusjuhendit algusest lõpuni läbi töötama.

Iga tehtud tarne saab koodihoidlasse laadimisel repositooriumi poolt unikaalse identifikaatori – arendaja kohustus on lisada identifikaator tarne kirjeldusse, selle põhjal võib ülalhoid veenduda mis versioon rakendusest koostamist vajab. Vastavalt identifikaatorile saab ülalhoid lähtekoodi alla laadida ehitusserverisse.

Vastavalt nõutele peab iga paigaldatav Java rakendus saama pakendatud artifaktiks, olenevalt infosüsteemist kas .ear või .war failiks. Nende loomiseks luuakse igale lähtekoodis olevale rakendusele vastav kompileerimise plaan – rida samme mille lõpptulemuseks on pakendatud artifakt. Plaani jooksutamise käigus paigaldatakse keskkonnapõhised seadistused pakendatava rakenduse sisse, selle tulemusena tekitab pidevintegratsioonivahend lähtekoodist kaks kuni neli erineva seadistusega versiooni antud rakendusest (vastavalt keskkondade arvule). Kompileerimise protsess toimub vastavalt projektile Maven või Gradle tööriistadega mis on pidevintegratsioonivahendisse sisse ehitatud. Rakendused mille seadistused loetakse sisse rakendusserveri kettalt ei vaja pidevintegratsioonis mitmekordset kompileerimist, kuid nende kasutusega kaasnevad riskid – seadistused pole tsentraalselt hallatavad ning rikete või hooldustööde korral

võivad need kustuda. Seadistuste kirjutamisega tegelevad süsteemiadministraatorid kes üle SSH protokolli tekitavad või redigeerivad seadistusfaile kas pidevintegratsiooniserveri või rakendusserverite ketastel.

Peale rakenduse ehitamist paigaldatakse ja käivitatakse rakendus süsteemiadministraatorite poolt. Vastavalt paigaldusjuhendile kirjeldab administraator rakenduse asukoha ning serveeritavad aadressid koormusjaoturis/http serveris, peale mida muutub rakendus vastavalt spetsifikatsioonile nähtavaks – sõltuvalt kas teiste rakenduste, asutuse endi töötajate, klientide või väliste kasutajate jaoks. Süsteemiadministraatori manuaalne töö, kombineeritud pidevintegratsiooni ning paigaldusserveri järgi ootamisega kestab keskmiselt 15 minutit, kuid sellele eelneb ooteaeg mis sõltuvalt töökoormusest, käimasolevatest ülesannetest ja tarne tähtsusest võib kesta kuni 48h. Antud töös hindab autor keskmiseks ooteajaks 4h.

Kord kui süsteemiadministraator on veendunud et rakendusserver on veebiserveris töötavas olekus ning töö seadistuste ning koormusjaoturiga on lõppenud markeerib ta paigaldatud tarne tööde haldusvahendis paigaldatuks ning teavitab seeläbi projektijuhti. Sõltuvalt haldusalast või teenusest tekitab projektijuht ülesande rakendust testida kas rakendusadministraatoril või välispartneril – automaattestimisega hetkel asutuses ei tegeleta. Küll aga on hangetesse sisse kirjutatud nõue kolmandatele osapooltele automaatteste rakendustesse sisse kirjutada. Sõltuvalt keskkonnast ja rakendusest võivad automaattestid olla sisse või välja lülitatud – seda kontrollib ülalhoid Gradle või Maven kompileerimistööriistade läbi. Automaattestide ebaõnnestumisel reeglina kompileerimine samuti ebaõnnestub.

Rakenduse seire eest vastutab rakendusadministraator – vastavalt arendaja poolt loodud rakenduse seirelehtedele kaardistab rakendusadministraator need asutuse monitooringuvahendis ning lisab need enda tiimi ning vajadusel ka helpdeski seireekraanidele.

2.3 Protsessi optimeerimise vajadus

Kasutades SIPOC mudelit ning väärtusvoogude kaardistust paistab autorile et olemasolevas protsessis oleks võimalik hoida läbi automatiseerimise kokku märkimisväärne hulk aega mida seni on raisatud spetsialiste taga ootamise ja manuaalse

töö peale. Hinnanguliselt on iga tarne paigalduse alla raisatud neli tundi aega mille asemel saaks rakendust testida, arendajale tagasisidet anda ning viimane omakorda saaks vastavalt tagasisidele kas vigu parandada või uut funktsionaalsust lisada. Arvestades tarnete mahtu hoiaks ettevõtte tänu automatiseerimisele iga projekti raames kokku nädalaid raisatud aega.

Vastavalt klientide kasvavatele nõudmistele pole võimalik rakenduste koostamise sammudega jätkata viisil mis hõlmavad süsteemiadministraatorite puhul tavaks olnud manuaalselt tehtavaid operatsioone – rakenduste arv ning selle jätkuv kasv ei tee võimalikuks jätkata sarnase töökorralduse ning inimressursiga. Kasutades pidevintegratsiooni ja automatiseerimisvahendeid tuleb ühtlustada rakenduste koostamise plaanid ning luua automaatika kuidas infosüsteemide ehitamine, seadistamine, paigaldamine ja vastavate teavituste edastamine osapooltele toimuks sujuvamalt. Autor võtab eesmärgiks viia infosüsteemide tarnete protsessid seisu kus 90% arendajate poolt loodavatest muudatustest läbiks tarnehala 100% automaatselt ning samas omaks kõik vajalikud osapooled toimuvast ülevaadet ning oleks kaasatud protsessi.

Kasutades punktis 1.2.2 analüüsitud väärtusvoo kaardistust loob autor tabeli defineerimaks protsessi senist efektiivsust.

Tabel 1. Protsessi efektiivsuse hindamine

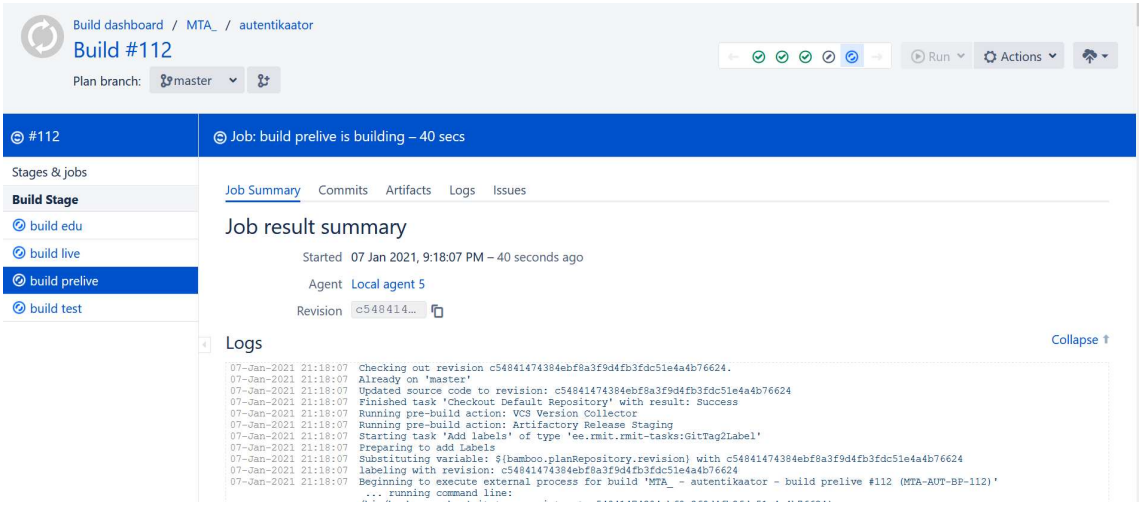
Protsessi etapid	**Efektiivsuse näitaja väärtus**
Ooteaeg (min)	480
Masinaeg (min)	27
Tsükli aeg (min)	10
Protsessi tsükli efektiivsus	7,15%

3 Tarneahela optimeerimine Autentikaatori ja Riigi Kinnisvararegistri näitel

3.1 Kompileerimine

Rakenduse ehitamise esimeseks võimalikuks sammuks mida parandada on pidevintegratsioonivahendi kasutamine viisil mis võimaldab jooksutada olemasolevaid koostamise plaane automaatselt. Peamine takistus selleks on olnud protseduuriline barjäär ülalhoiu ja arenduse meeskondade vahel – automaatika kasutamise eelduseks on kokkulepe millisest lähtekoodihoidla harusse rakendust tarnitakse (pidevintegratsioonivahendi tuge dünaamilistele harudele on piiratud ning seab omad piirangud ka edasisele lahendusele punktis 9.3). Selle parandamiseks soovitab autor lähtekoodihoidlas staatilise haru kasutamise lisada rakenduste hanke mittefunktsionaalsetesse nõuetesse.

Lisaks automaatsele kompileerimisele on võimalik parandada rakenduste kokku ehitamise aega projektide puhul mille seadistused kirjutatakse otse rakenduse sisse. Hetkel teostab pidevintegratsiooni ehituse plaan kompileerimise ülesande käigus kuni neli erineva seadistusega rakendust (sõltuvalt keskkondade arvust), raisates protsessi käigus aega. Kui aga kompileerimise plaani ülesanded jagada ära ehitusserverite agentide vahel võimaldab see märkimisväärset kokkuhoidu rakenduse ehitusel.



Joonis 7. Nelja erineva keskkonna kompileerimine pidevintegratsioonis

Autori poolt loodud keskkonnapõhine pidevintegratsiooni plaan rakenduste kompileerimiseks asub Lisa2.

3.2 Seadistamine

Iga rakenduse seadistamine on paratamatult suures osas käsitöö – rakenduse arendajal ei ole infot keskkondade spetsiifikast ning süsteemide hooldusel pole võimalik automatiseerida seadistusi ilma neist ette teadmata. Küll aga võib parandada manuaalset protsessi automaatika abil kasutades seadistuses väärtuseid mida muuta pole vaja : rakenduse arendaja on lähtekoodi lisanud seadistuste näite arenduses töötavast rakendusest – leppides arendajaga kokku selle asukohas saab pidevintegratsiooni kaudu automaatselt vahetada seadistustes väärtused mis on keskkonnapõhised. Seeläbi ei pea lisandunud seadistused mida arendaja on osanud õigesti väärtustada kaasa tooma muudatusi ülalhoiu enda seadistustes – vajalikud väärtused asendatakse pidevintegratsioonivahendis vastavalt kas ehituse (seadistused on rakendusse sisse ehitatud) või paigalduse (seadistused loetakse rakendusserveri kettalt) hetkel.

Autori poolt loodud pidevintegratsioon plaan mille käigus toimub seadistuste vahetus asub Lisa 3. Seadistuste redigeerimiseks kasutab autor vastavat Linux käsku [13].

3.3 Paigaldamine

Rakenduste paigalduste automatiseerimiseks tuleb esmalt leida viis kuidas eraldada tarned mida on võimalik automaatikaga paigalda nendest mis vajavad eelnevat käsitsi ümber seadistamist. Ülalhoiul antud informatsiooni pole, küll aga rakenduse tarnijal (arendajal). Kord kui arendaja lisab õigesti väärtustatud muutuja vastavalt kokku lepitud faili lähtekoodihoidlas on tarnete eraldamine võimalik.

Teine kitsaskoht mida lahendada on rakenduste defineerimine versiooninumbri ja ehituse järjekorranumbriga pidevintegratsioonis ja lähtekoodihoidlas – seni kuni iga paigaldus markeeritakse peale ehitamist arendaja poolt edastatud versiooniga käsitöö käigus süsteemiadministraatorite poolt on tagatud järjepidevus ning ülevaade mis versioon rakendusest mis keskkonda paigaldatud on. Sarnaselt esimesele punktile saab seda lahendada protseduurilisel teel – arendaja peab edastama info kuhu rakenduse tarneinfot kirjutatakse ning mis muutuja viitab versioonile. Vastavalt sellele saab pidevintegratsioonivahendi plaani ülesande osa antud muutuja sisse lugeda ning peale rakenduse kompileerimist sisemiselt korrektselt märgistada.

Kord kui arendajatel on võimalus nõuetele vastavaid tarneid vastavalt markeerida ning on tagatud publitseerimise järjepidevus saab luua lahenduse pidevintegratsiooni paigaldusserveri poolel. Selleks tuleb lisaks rakendusele ning baasimuudatustele defineerida fail vastavate muutujatega artifakti näol mida pidevintegratsioonitööriist saab liigutada ehitusserveri ja paigaldusserveri vahel. Sellele järgmiseks sammuks oleks ehitada rakenduse paigalduse eelduse kontroll ning siduda see baasimuudatuste ja rakenduse paigalduse käivitusega.

Töövahendi kaudu on võimalik saata asjaosalistele ka automaatseid teavitusi rakenduse paigalduste sammude staatuste kohta, kuid infoturbest tulevate piirangute tõttu puudub võime edastada väliste arendajatele detailset infot pidevintegratsioonis toimuvast.

Autor poolt loodud paigalduse võtme import, rakenduste versiooni lugemine, paigalduse eelduse kontroll ning vastav käivitus tingimustele vastamisel on pidevintegratsiooni plaani seadistuses Lisa 3. Teostamaks väärtuse kontrolli kasutab autor vahesammu vastava Linuxi käskudega awk [17] ja echo [18].

3.4 Testimine

Asutuse infosüsteemide hangetes puudub nõue automaatteste tarnida ning asutuses endal puudub ka võimekus vastavaid teste kirjutada. Vastavate testide olemasolul saavad arendajad nende sõltuvuse sisse kirjutada rakenduse kompileerimise plaani ning pidevintegratsiooni saab kirjeldada testidest sõltuvuse.

3.5 Seire

Rakenduse seire automatiseerimiseks pakub autor lahendust mis hõlmab punktis 7.3 kirjeldatud koormusjaoturi ja http serveri seadistuste automaatse laadimise lahenduse (teenuste registri) laiendust. Defineerides lisaks rakenduse enda asukohale ka seirelehtede asukohad läbi teenuste registri automaatika tekib võimalus lisada arendaja poolt lisatud sirelehed monitooringuvahendisse ilma käsitööta. Otsest tehnilist lahendust seirega ei kaasne, kuid läbi kommunikatsiooni ning mittefunktsionaalsete nõuete täiendamise on võimalik saavutada märkimisväärne ajaline kokkuhoid ning vähendada monitooringu eest vastutavate rakendusadministraatorite tööd.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli uurida võimalusi kuidas avaliku sektori asutuse veebirakenduste tarnetega seonduvaid protsesse automatiseerida - autor uuris võimalusi kuidas oleks võimalik kasutada olemasolevaid töövahendeid efektiivsemalt, vähendada manuaalset tööd ning elimineerida protseduurilisi pudelikaelu. Uurimise all olid rakenduste ehituse, seadistamise, paigaldamise, testimise ja seirega seotud protsessid. Autor kasutas protsesside aegluse analüüsimiseks väärtusvoo kaardistust ning seejärel leidis lahendusi kuidas vajalikkus manuaalse töö järele kaotada.

Käsitletavateks probleemideks olid protsesside aeglus ning asutuse süsteemide ülalhoiu meeskonna poolt tehtav liigne sõltuvus manuaalsest tööst mis põhjustas rakenduste tarnimisel pudelikaelade tekkimist. Liikumises monoliitse rakenduste arhitektuuri pealt mikroteenustele tekkis olukord kus olemasolev töökorraldus poleks olnud enam jätkusuutlik.

Probleemi lahendamiseks leidis autor viise kuidas olemasolevate töövahendite täiendamisel saab kõiki rakenduste tarnimisahela protsesse automatiseerida ning vähendada käsitööd jooksvate tarnete puhul sisuliselt miinimumini. Andes arendajatele rohkem vastutust ja tagasisidet ning võimaldades omakorda kiiremini veebirakendusi tarnida tekkis võimalus tarnida tarkvara senisest märgatavalt tihedamalt, saada ühtlasi kiiret tagasisidet tarnetes sisalduvate funktsioonide ja/või veaparanduste kohta. Ülalhoiu manuaalset sekkumist vajavad küll uued lisanduvad teenused või tarned mille käigus tuleb teha muudatusi platvormi kihil, kuid hoolimata sellest ei vaja orienteeruvalt 90% rakenduste tarnetest enam manuaalset tööd ega tekita ahelates pudelikaelu. Püstitatud ülesande loeb autor täidetuks, kuid leiab et tulevikus on võimalik automatiseerimist veel enamgi täiendada läbi automaatse platvormide või konteinerite provisioneerimiste.

Kasutatud kirjandus

[1] „State of Devops 2019“, Devops Research and Assesment (DOSA) 2019 [Võrgumaterjal]. URL: https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

Loetud 15.11.2020

[2] „Why Great Product Companies Release Software to Production Multiple Times a Day“, Devops Zone 2018 [Võrgumaterjal]. URL: https://dzone.com/articles/why-do-great-product-companies-release-software-to

Loetud 12.11.2020

[3] „Devops Fundamentals“, Devops Agile Skills Association 2017

Loetud 04.11.2020

[4] „DASA Devops Fundamentals Syllabus“, Devops Agile Skills Association 2017

Loetud 04.11.2020

[5] „Software teams spend one day a week, and more, troubleshooting code issues“, Joe McKendrick 2020 [Võrgumaterjal]. URL: https://www.zdnet.com/article/software-teams-spend-one-day-a-week-and-more-troubleshooting-code-issues/

Loetud 04.01.2021

[6] „A Different Meaning of CI - Continuous Improvement, the Heartbeat of DevOps“, Sabine Wojcieszak 2019 [Võrgumaterjal]. URL: https://www.infoq.com/articles/continuous-improvement-heartbeat-devops/

Loetud 04.01.2021

[7] „Embracing digital disruption by adopting DevOps practices“, Quint Wellington Redwood 2017 [Võrgumaterjal]. URL: https://www.quintgroup.com/en-in/insights/white-paper-embracing-digital-disruption-by-adopting-devops-practices/

Loetud 04.01.2021

[8] „What Is Etcd and How Do You Set Up an Etcd Kubernetes Cluster?“, Calin Rus 2019 [Võrgumaterjal]. URL: https://rancher.com/blog/2019/2019-01-29-what-is-etcd/

Loetud 05.01.2021

[9] „How To Use Confd and Etcd to Dynamically Reconfigure Services in CoreOS“ Justin Ellingwood 2014 [Võrgumaterjal]. URL: https://www.digitalocean.com/community/tutorials/how-to-use-confd-and-etcd-to-dynamically-reconfigure-services-in-coreos

Loetud 28.11.2020

[10] „The Role of Continuous Delivery in IT and Organizational Performance“, Nicole Forgsen, Jez Humble 2016 [Võrgumaterjal]. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2681909

Loetud 18.11.2020

[11] „Let’s Talk about Value Stream Mapping with Respect to DevOps“, Surinderpal S. Kumar 2019 [Võrgumaterjal]. URL: https://www.addteq.com/blog/2019/09/lets-talk-about-value-stream-mapping-with-respect-to-devops

Loetud 28.11.2020

[12] „What is value stream mapping?“, Juni Mukherjee [Võrgumaterjal]. URL: https://www.atlassian.com/continuous-delivery/principles/value-stream-mapping

Loetud 02.12.2020

[13] „How to use sed to find and replace text in files in Linux / Unix shell“, Vivek Gite 2018 [Võrgumaterjal]. URL: https://www.cyberciti.biz/faq/how-to-use-sed-to-find-and-replace-text-in-files-in-linux-unix-shell/

Loetud 02.12.2020

[14] „Secure Shell (SSH)“, Margaret Rouse [Võrgumaterjal] URL: https://searchsecurity.techtarget.com/definition/Secure-Shell

Loetud 28.11.2020

[15] „HTTP Meaning & Definition“, Abby Dykes [Võrgumaterjal] URL: https://www.webopedia.com/definitions/http/

Loetud 28.11.2020

[16] „What is Continuous Integration?“, Atlassian [Võrgumaterjal]. URL: https://www.atlassian.com/continuous-delivery/continuous-integration

Loetud 28.11.2020

[17] „AWK command in Unix/Linux with examples“, GeeksforGeeks 2019 [Võrgumaterjal]. URL: https://www.geeksforgeeks.org/awk-command-unixlinux-examples/

Loetud 04.01.2021

[18] „Echo Command in Linux with Examples“, Linuxize 2019 [Võrguamterjal]. URL: https://linuxize.com/post/echo-command-in-linux-with-examples/

Loetud 04.01.2021

[19] „What is an artifact?“, Jetbrains 2020 [Võrgumaterjal]. URL: https://www.jetbrains.com/help/idea/working-with-artifacts.html

Loetud 07.01.2021

[20] „Lean IT Foundation Participant Handbook“, Quint Wellington Redwood 2019

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Margus Koval

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebirakenduste koostamine Java keskkonnas avaliku sektori ettevõtte näitel“, mille juhendaja on Vladimir Viies
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

04.01.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Pidevintegratsiooni plaani seadistus rakenduse ehitamiseks

```
version: 2
plan:
  project-key: MTA
  key: AUT
  name: autentikaator
stages:
- Build Stage:
    manual: false
    final: false
    jobs:
    - build prelive
    - build live
    - build edu
    - build test
build prelive:
  key: BP
#
# Some plugin configurations are not supported by YAML Specs
  tasks:
  - checkout:
      force-clean-build: 'false'
# Task ee.rmit.rmit-tasks:GitTag2Label is not supported yet
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
      - |-
        cp /opt/atlassian/bamboo-
conf/MTA/authenticator/prelive/authenticator.properties config/dev
        cp /opt/atlassian/bamboo-conf/MTA/authenticator/prelive/logback.xml
src/main/resources/logback.xml
  - maven:
      executable: maven-3.2.5
      jdk: JDK 1.6
      goal: clean package -P dev
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
      - |-
        if [ -d prelivetarget ] ; then
        echo "prelivetarget exists"
        else
        mkdir prelivetarget
        fi
        mv target/*.war prelivetarget/authenticator.war
  artifacts:
  - name: prelive_war
```

```
    location: prelivetarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: kool_war
    location: kooltarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: findbugs
    location: sandbox/findbugs
    pattern: '*.html'
    shared: false
    required: false
  - name: test_war
    location: testtarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: dev_war
    location: devtarget
    pattern: '*.war'
    shared: false
    required: false
  - name: prod_war
    location: prodtarget
    pattern: authenticator.war
    shared: false
    required: false
build live:
  key: BL
#
# Some plugin configurations are not supported by YAML Specs
  tasks:
  - checkout:
      force-clean-build: 'false'
# Task ee.rmit.rmit-tasks:GitTag2Label is not supported yet
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
      - cp /opt/atlassian/bamboo-
conf/MTA/authenticator/live/authenticator.properties config/prod
  - maven:
      executable: maven-3.2.5
      jdk: JDK 1.6
      goal: clean package -P prod
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
      - |
        if [ -d prodtarget ] ; then
```

```
        echo "prodtarget exists"
        else
        mkdir prodtarget
        fi
        mv target/*.war prodtarget/authenticator.war
  artifacts:
  - name: prelive_war
    location: prelivetarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: kool_war
    location: kooltarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: findbugs
    location: sandbox/findbugs
    pattern: '*.html'
    shared: false
    required: false
  - name: test_war
    location: testtarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: dev_war
    location: devtarget
    pattern: '*.war'
    shared: false
    required: false
  - name: prod_war
    location: prodtarget
    pattern: authenticator.war
    shared: false
    required: false
build edu:
  key: JOB1
#
# Some plugin configurations are not supported by YAML Specs
  tasks:
  - checkout:
      force-clean-build: 'false'
# Task ee.rmit.rmit-tasks:GitTag2Label is not supported yet
  - maven:
      executable: maven-3.2.5
      jdk: JDK 1.6
      goal: clean package -P dev
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
```

```
      - |-
        if [ -d kooltarget ] ; then
        echo "kooltarget exists"
        else
        mkdir kooltarget
        fi
        mv target/*.war kooltarget/authenticator.war
  artifacts:
  - name: prod_war
    location: prodtarget
    pattern: authenticator.war
    shared: true
    required: false
  - name: dev_war
    location: devtarget
    pattern: '*.war'
    shared: true
    required: false
  - name: test_war
    location: testtarget
    pattern: authenticator.war
    shared: true
    required: false
  - name: findbugs
    location: sandbox/findbugs
    pattern: '*.html'
    shared: false
    required: false
  - name: kool_war
    location: kooltarget
    pattern: authenticator.war
    shared: true
    required: false
  - name: prelive_war
    location: prelivetarget
    pattern: authenticator.war
    shared: true
    required: false
build test:
  key: BT
#
# Some plugin configurations are not supported by YAML Specs
  tasks:
  - checkout:
      force-clean-build: 'false'
# Task ee.rmit.rmit-tasks:GitTag2Label is not supported yet
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
      - |-
```

```
        cp /opt/atlassian/bamboo-
conf/MTA/authenticator/test/authenticator.properties config/test
        cp /opt/atlassian/bamboo-conf/MTA/authenticator/test/logback.xml
src/main/resources
  - maven:
      executable: maven-3.2.5
      jdk: JDK 1.6
      goal: clean package -P test
  - script:
      interpreter: BINSH_OR_CMDEXE
      scripts:
      - |-
        if [ -d testtarget ] ; then
        echo "testtarget exists"
        else
        mkdir testtarget
        fi
        mv target/*.war testtarget/authenticator.war
  artifacts:
  - name: prelive_war
    location: prelivetarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: kool_war
    location: kooltarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: findbugs
    location: sandbox/findbugs
    pattern: '*.html'
    shared: false
    required: false
  - name: test_war
    location: testtarget
    pattern: authenticator.war
    shared: false
    required: false
  - name: dev_war
    location: devtarget
    pattern: '*.war'
    shared: false
    required: false
  - name: prod_war
    location: prodtarget
    pattern: authenticator.war
    shared: false
    required: false
triggers:
- polling: 180
```

```
branches:
  create: manually
  delete: never
  link-to-jira: true
notifications: []
labels: []
other:
  concurrent-build-plugin: system-default
---
version: 2
plan:
  key: MTA-AUT
plan-permissions:
- groups:
  - developers
  permissions:
  - view
  - edit
  - build
- roles:
  - logged-in
  - anonymous
  permissions:
  - view
- users:
  - admin
  permissions:
  - view
  - edit
  - build
  - clone
  - admin
```

Lisa 3 – rakenduse automaatne paigaldus ja selle kontroll, teavitused, ehituse-ja versiooninumbri publitseerimine ning seadistuste laadimine koodihoidlast

```
version: 2
plan:
  project-key: RM
  key: KO
  name: KVR opensource
stages:
- Default Stage:
    manual: false
    final: false
    jobs:
    - Default Job
Default Job:
  key: JOB1
  tasks:
  - checkout:
      force-clean-build: 'true'
# Task ee.rmit.rmit-tasks:GitTag2Label is not supported yet
  - script:
      interpreter: SHELL
      scripts:
      - |-
        #cp /opt/atlassian/bamboo-conf/RM/rkvr/pom.xml rkvr-
backend/db/pom.xml
        #cp /opt/atlassian/bamboo-conf/RM/rkvr/test/package.json rkvr-
frontend/package.json
        cp /opt/atlassian/bamboo-conf/RM/rkvr/test/rkvr-frontend-
properties.js rkvr-frontend/public/rkvr-frontend-properties.js
        sed -i -e 's/FEPLACEHOLDER/rkvr-frontend/g' rkvr-
frontend/package.json
  - maven:
      executable: maven-3.6.3
      jdk: openjdk-11
      goal: clean package -DskipTests -Dhttps.proxyHost=cache1.rmv -
Dhttps.proxyPort=3128 -Dhttps.nonProxyHosts=localhost|*.rmv
  - script:
      interpreter: SHELL
      scripts:
      - |-
        #cp /opt/atlassian/bamboo-conf/RM/kvr/pom.xml rkvr-backend/db/pom.xml
        mv rkvr-backend/target/rkvr-*.war rkvr-backend/target/rkvr.war
# Task com.davidehringer.atlassian.bamboo.maven.maven-pom-parser-
plugin:maven-pom-parser-plugin is not supported yet
  - script:
      interpreter: SHELL
```

```
      scripts:
      - |-
        touch versioon.properties
        echo "prop=`printf "%05d" ${bamboo.buildNumber}`" >
versioon.properties
  - inject-variables:
      file: versioon.properties
      scope: RESULT
      namespace: versioon
  artifacts:
  - name: autodeploy
    location: .
    pattern: autodeploy.properties
    shared: true
    required: false
  - name: db
    location: rkvr-backend/db/
    pattern: '**/*'
    shared: true
    required: true
  - name: rkvr.war
    location: rkvr-backend/target/
    pattern: rkvr.war
    shared: true
    required: false
  - name: rkvr-frontend.war
    location: rkvr-frontend/target/
    pattern: rkvr-frontend.war
    shared: true
    required: true
triggers:
- polling: 180
branches:
  create: manually
  delete: never
  link-to-jira: true
notifications: []
labels: []
other:
  concurrent-build-plugin: system-default
---
version: 2
plan:
  key: RM-KO
plan-permissions:
- users:
  - margus.koval
  permissions:
  - view
  - edit
  - build
```

```
  - clone
  - admin

version: 2
deployment:
  name: KVR opensource
  source-plan: RM-KO
release-naming:
  next-version-name: ${bamboo.maven.version}#${bamboo.versioon.prop}
  applies-to-branches: true
  auto-increment: false
  auto-increment-variables: []
environments:
- TEST LIQUIBASE
- TEST BACKEND
- TEST AUTODEPLOY CHECK
- TEST FRONTEND
TEST LIQUIBASE:
  triggers:
  - environment-success: TEST AUTODEPLOY CHECK
  tasks:
  - clean
  - artifact-download:
      artifacts:
      - name: db
  - script:
      interpreter: SHELL
      scripts:
      - cp /opt/atlassian/bamboo-conf/RM/KVR/test/liquibase.properties .
  - maven:
      executable: maven-3.6.1
      jdk: JDK 11
      goal: liquibase:update
  final-tasks: []
  variables: {}
  requirements: []
  notifications:
  - events:
    - deployment-failed
    recipients:
    - emails:
      - arendaja1@arendaja.com
    - emails:
      - arendaja2@arendaja.com
  - events:
    - deployment-started-and-finished
    recipients:
    - emails:
      - arendaja1@arendaja.com
    - emails:
      - arendaja2@arendaja.com
```

```
TEST BACKEND:
  triggers:
  - environment-success: TEST LIQUIBASE
  tasks:
  - clean
  - artifact-download:
      artifacts:
      - name: rkvr.war
# Task com.atlassian.bamboo.plugins.tomcat.bamboo-tomcat-plugin:deployAppTask
is not supported yet
# Task com.atlassian.bamboo.plugins.tomcat.bamboo-tomcat-plugin:deployAppTask
is not supported yet
  final-tasks: []
  variables: {}
  requirements: []
  notifications:
  - events:
    - deployment-failed
    recipients:
    - emails:
      - arendaja1@arendaja.com
    - emails:
      - arendaja2@arendaja.com
  - events:
    - deployment-started-and-finished
    recipients:
    - emails:
      - arendaja1@arendaja.com
    - emails:
      - arendaja2@arendaja.com
TEST AUTODEPLOY CHECK:
  triggers:
  - build-success
  tasks:
  - clean
  - artifact-download:
      artifacts:
      - name: autodeploy
  - inject-variables:
      file: autodeploy.properties
      scope: LOCAL
      namespace: autodeploy
  - script:
      interpreter: SHELL
      scripts:
      - |-
        awk -vs1="true" -vs2="${bamboo.autodeploy.TEST}" 'BEGIN {
          if ( tolower(s1) == tolower(s2) ){
            echo "auto deploy is set"
          }
          else {
```

```
            echo "autodeploy is ${bamboo.autodeploy.TEST}"
            exit 1 }
        }'
  final-tasks: []
  variables: {}
  requirements: []
  notifications: []
TEST FRONTEND:
  triggers:
  - environment-success: TEST LIQUIBASE
  tasks:
  - clean
  - artifact-download:
      artifacts:
      - name: rkvr-frontend.war
# com.atlassian.bamboo.plugins.scripttask:task.builder.script is disabled.
This state is not supported at YAML
# Task com.atlassian.bamboo.plugins.tomcat.bamboo-tomcat-plugin:deployAppTask
is not supported yet
# Task com.atlassian.bamboo.plugins.tomcat.bamboo-tomcat-plugin:deployAppTask
is not supported yet
  final-tasks: []
  variables: {}
  requirements: []
  notifications:
  - events:
    - deployment-failed
    recipients:
    - emails:
      - arendaja1@arendaja.com
    - emails:
      - arendaja2@arendaja.com
  - events:
    - deployment-started-and-finished
    recipients:
    - emails:
      - arendaja1@arendaja.com
    - emails:
      - arendaja2@arendaja.com
---
version: 2
deployment:
  name: KVR opensource
deployment-permissions:
- users:
  - margus.koval
  permissions:
  - view
  - edit
environment-permissions:
- TEST LIQUIBASE:
```

```
  - users:
    - margus.koval
    permissions:
    - view
    - edit
    - deploy
- TEST BACKEND:
  - users:
    - margus.koval
    permissions:
    - view
    - edit
    - deploy
- TEST AUTODEPLOY CHECK:
  - users:
    - margus.koval
    permissions:
    - view
    - edit
    - deploy
- TEST FRONTEND:
  - users:
    - admin
    permissions:
    - view
    - edit
    - deploy
...
```