SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

JATKUVA INTEGRAATIO JA TOIMITUS/JULKAISU CASE KIHO

TEKIJÄ/T: Petteri Huotari

SAVONIA-AMMATTIKORKEAKOULU OPINNÄYTETYÖ

Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala			
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma			
Työn tekijä(t) Petteri Huotari			
Työn nimi Jatkuva integraatio ja toimitus/julkaisu case Kiho			
Päiväys	15.4.2020	Sivumäärä/Liitteet	29
Ohjaaja(t) Keijo Kuosmanen, Pasi Liimatainen			
Toimeksiantaja/Yhteistyökumppani(t) Mastercom / Kiho Oy			

Tiivistelmä

Tämän opinnäytetyön aiheena oli jatkuvan integraation ja toimituksen sekä julkaisun (CI/CD) järjestelmän lisääminen osaksi Kihon ohjelmistokehitystä tehostamaan tuotekehitystä ja nopeuttamaan uusien ominaisuuksien toimittamista asiakkaille.

Opinnäytetyössä tehtiin vertailu erilaisille automaatiopalvelimille ja palveluille. Tehdyn vertailun pohjalta valittiin käytettäväksi automaatiopalvelimeksi Jenkins. Opinnäytetyössä kehitettiin Jenkinsin avulla CI/CD järjestelmä, joka kääntäisi ja testaisi Kihon GPSd v2 palvelimen. Järjestelmä myös julkaisi GPSd v2:n testausympäristöön tarvittaessa.

Opinnäytetyön lopputuloksena saatiin järjestelmä, joka täyttää toimeksiantajan sille alussa asettamat vaatimukset Järjestelmän pohjalta on tarkoitus rakentaa CI/CD järjestelmät muillekin Kihon ohjelmistoprojekteille.

Avainsanat
Jatkuva integraatio, Jatkuva toimitus, Jatkuva julkaisu

SAVONIA UNIVERSITY OF APPLIED SCIENCES THESIS

Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Computer Science			
Author(s) Petteri Huotari			
Title of Thesis Continuous Integration and Delivery/Deployment in Case Kiho			
Date	15.4.2020	Pages/Appendices	29
Supervisor(s) Keijo Kuosmanen, Pasi Liimatainen			
Client Organization /Partners Mastercom / Kiho Oy			

Abstract

The purpose of this thesis was the integration of continuous integration and delivery as well as the release of the (CI / CD) system into Kiho's software development to enhance product development and accelerate delivery of new features to customers.

The thesis compared different automation servers and services. Based on this comparison, Jenkins was selected as the automation server to be used. Jenkins was used to develop a CI / CD system to compile and test Kiho's GPSd v2 server. The system also released GPSd v2 for the testing environment if necessary.

The end result is a CI/CD system that fulfills all the requirements given by the client, and the basis of the system can be used in the future as a core for more CI/CD systems in other development projects of Kiho.

Keywords
Continuous integration, Continuous delivery, Continuous deployment

SISÄLLYS

JOHDANTO

Opinnäytetyön toimeksiantajana toimi Mastercom / Kiho Oy. Yritys on Kuopiossa sijaitseva vuonna 2003 perustettu ohjelmistoalan yritys, joka tuottaa asiakkailleen työajanseurannan, kalustonhallinnan ja GPS-paikannuksen palveluita. Toimeksiantajalla oli tarve ottaa käyttöön jatkuvan integraation ja toimituksen sekä julkaisun järjestelmä tehostamaan tuotekehitystä ja nopeuttamaan uusien ominaisuuksien toimittamista asiakkaille. Työn tavoitteena oli löytää toimeksiantajalle paras ja kustannustehokkain tapa liittää jatkuvan integraation ja toimituksen/julkaisun järjestelmä toimeksiantajan ohjelmistokehitykseen.

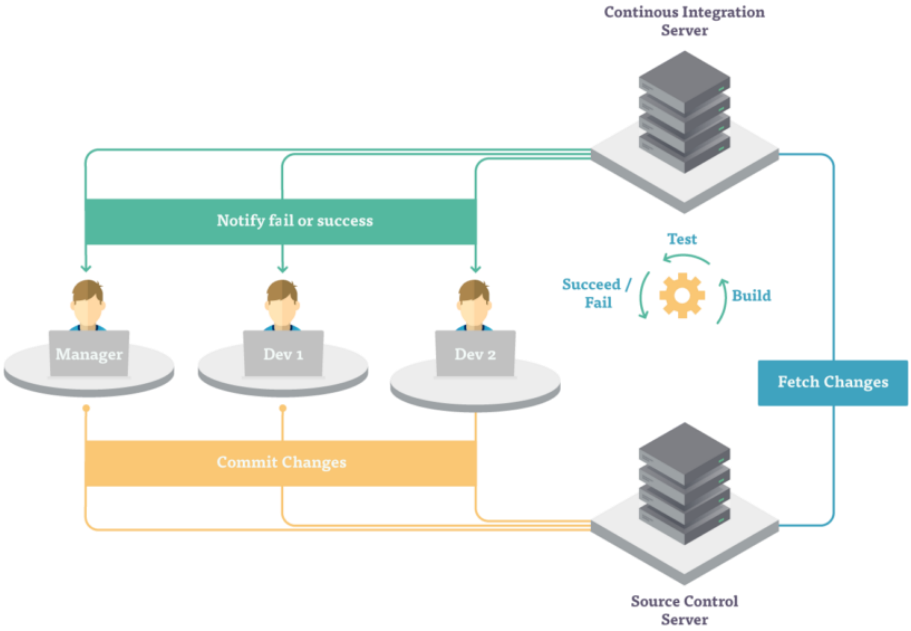
Tämän opinnäytetyön ensimmäisessä luvussa käydään läpi, mitä tarkoitetaan jatkuvalla integraatiolla ja toimituksella sekä julkaisulla, sekä käydään läpi siihen liittyviä käsiteitä. Tämän jälkeen käsitellään jatkuvan integraatioon sekä jatkuvan toimitukseen että käyttöönottoon tarvittavia työkaluja sekä vertaillaan saatavilla olevien työkalujen hyviä ja huonoja puolia. Sitten tutustutaan erilaisiin automaatio palvelimiin, sekä vertailemaan niiden eroavaisuuksia sekä ominaisuuksia. Tämän jälkeen käsitellään tämän opinnäytetyön tekemisessä olleita vaiheita sekä sen toimeksiantajalle tuottamaa lopputulosta. Opinnäytetyön vaiheiden esittelyn jälkeen esitellään lopuksi omia pohdintoja työn aikana tehdyistä valinnoista, sen vaiheista, sekä myös lopputuloksesta.

1 YLEISTÄ AIHEESTA

Tässä luvussa käydään läpi, mitä tarkoitetaan jatkuvalla integraatiolla (engl. Continuous Integration, CI) sekä siihen liittyvillä jatkuvalla toimituksella (engl. Continuous Delivery, CD) ja jatkuvalla julkaisulla (engl. Continuous Deployment, CD).

1.1 Jatkuva integraatio

Jatkuvalla integraatiolla tarkoitetaan ohjelmistokehitysmenetelmää, jossa kehittäjäryhmän jäsenet pyrkivät integroimaan tekemänsä koodimuutokset mahdollisimman usein versionhallinnan päähaaraan. Muutokset tulisi integroida vähintään kerran päivässä, jotta muutoksista johtuvat mahdolliset integraatio-ongelmat sekä bugit havaittaisiin ja pystyttäisiin korjaamaan mahdollisimman nopeasti. (Fowler, 2006; Pittet, a)

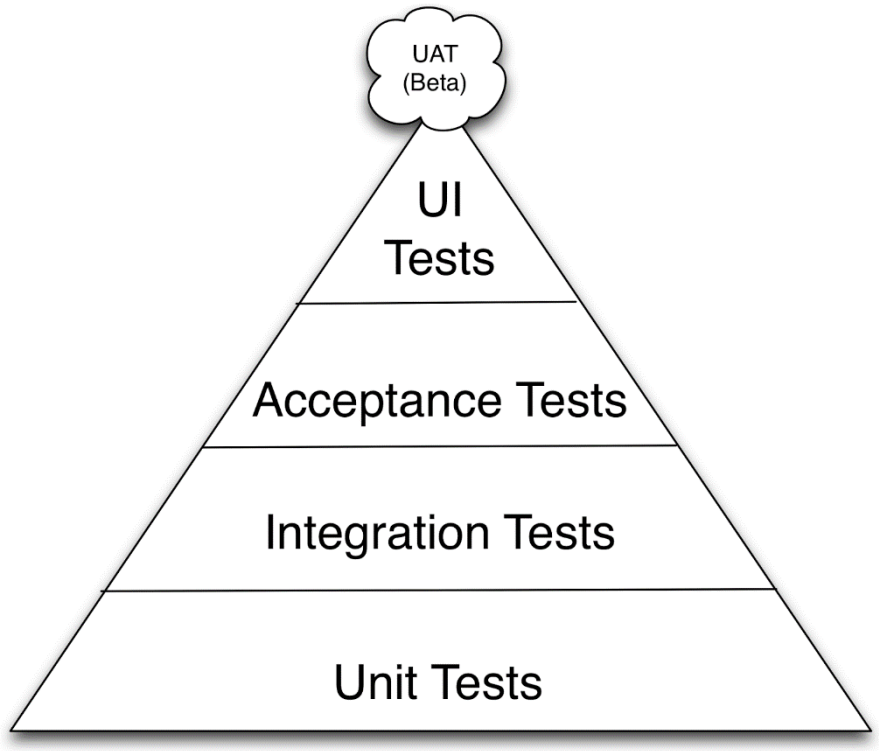


Kuva 1. Esimerkki jatkuvasta integraatioprosessista.

Ensimmäinen ja tärkein osa jatkuvaa integraatiota on koonnin sekä testauksen automatisointi vähintään jokaisen versionhallinnan päähaarassa tapahtuvan muutoksen jälkeen. Olisi kuitenkin suositeltavaa, että automaattinen koonti sekä testaus tehtäisiin kaikille versionhallinnassa oleville haaroille niissä tapahtuneiden muutosten jälkeen. Automatisoidussa koonnissa ja testauksessa ohjelmisto käännetään ja kootaan suorituskelpoiseksi ohjelmaksi sekä sille ajetaan useita erilaisia testejä. Testeillä pyritään varmistamaan vanhojen sekä uusien ominaisuuksien oikeanlainen toiminta sekä havaitsemaan integraation virheet ja bugit mahdollisimman aikaisessa vaiheessa. Jos testauksessa havaitaan ongelmia, tulisi niistä vastuussa olevan ohjelmistokehittäjän korjata virheet mahdollisimman nopeasti. (Fowler, 2006; Pittet, a)

Pittet'n mukaan testit voidaan jakaa neljään ryhmään.

- Yksikkötestit, joissa testataan yhden metodin tai funktion toimintaa.
- Integraatiotestit, joissa testataan luokkien ja funktioiden yhteistoimintaa.
- Hyväksyntätestit, joissa testataan käyttötapakuvauksien mukaista toimintaa.
- Käyttöliittymätestit, joissa testataan ohjelmiston toimintaa käyttäjän näkökulmasta. (Pittet, a)



Kuva 2. Pittet'n mukaan testejä tulisi painottaa kuvassa olevan pyramidin mukaan.

Automaattisista testeistä suurin osa tulisi siis olla yksikkötestejä, koska niitä on nopea sekä tehdä että ajaa. Niiden pohjalta löydettyjen bugien paikallistaminen koodista on helpointa. Pienin osa taas tulisi olla käyttöliittymätestejä, koska niiden ajaminen vie kauan sekä niiden tekeminen on monimutkaista. On tärkeää, että testit kattaisivat mahdollisimman suuren osan ohjelmiston ominaisuuksista ja testien hyvänä kattavuutena pidetäänkin noin 80 prosenttia koodeista. Kuitenkin testien kattavuutta tärkeämpänä ominaisuutena pidetään testien laatua sekä sitä että testit kohdistuisivat ohjelmiston toiminnan kannalta kriittisimpiin osiin. Uusia testejä tulisi kirjoittaa aina bugeja korjatessa sekä uusien ominaisuuksien ja koodin uudelleenkirjoituksen yhteydessä. (Pittet, a; Cohn, 2009)

Automaattisen koonnin ja testaamisen tulisi olla mahdollisimman nopeaa, mielellään alle kymmenen minuuttia. Tämä ei kuitenkaan ole aina mahdollista esimerkiksi, jos projekti on vanha, testejä on paljon tai jos niissä testataan järjestelmän suorituskykyä ja käyttäjäinteraktioita. Tällöin voidaan testit jakaa useampaan vaiheeseen ja raportoida kehittäjälle testien tuloksista sitä mukaa kun

testien ajaminen valmistuu. Tällöin kehittäjän ei tarvitse odottaa testien valmistumista jatkaakseen työskentelyä. (Fowler, 2006)

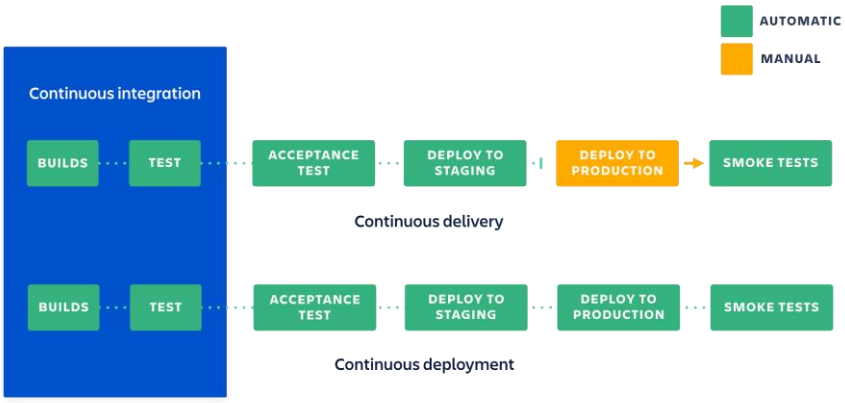
Toinen osa jatkuvaa integraatiota on kulttuurimuutos organisaatiossa tukemaan jatkuvan integraation käytäntöjä. Jatkuvaa integraation käytettäessä kehittäjien ei tulisi enää kehittää muutosta kokonaan valmiiksi omassa versionhallinnan haarassaan. Sen sijaan muutokset tulisi kehittää pienissä osissa, joita integroidaan usein päähaaraan. Kehittäjien tulisi myös kirjoittaa jokaiselle kehitetylle ominaisuudelle kattavat testit ohjelmiston käyttötapauskuvausten pohjalta. Kehittäjien kaikkein tärkeimpänä prioriteettina pitäisi myös olla päähaaran korjaaminen, jos siihen tehdyt muutokset rikkovat sen. (Pittet, a)

1.2 Jatkuva toimitus ja julkaisu

Jatkuvalla toimituksella tarkoitetaan ohjelmistokehitysmenetelmää, jossa kehittäjäryhmän jäsenet pyrkivät kehittämään ohjelmistoa siten että se on versionhallinnassa aina siinä tilassa, että se voitaisiin julkaista milloin vain. Jotta tähän päästäisiin tulisi muutosten olla pieniä ja ne tulisi julkaista usein, sillä jos muutoksissa ilmenee virheitä, ne olisivat helpompi ja nopeampi korjata. Jatkuvaa toimitusta käytetään yleensä jatkamaan jatkuvaa integraatiota. (Fowler, 2013)

Kuten jatkuvassa integraatiossakin jatkuvan toimituksen ensimmäinen ja tärkein osa on automaatio. Automaatioon tarkoituksena on automatisoida ohjelmiston siirtely testaus- sekä tuotantoympäristöjen välillä, jotta ohjelmiston julkaisusta tulisi helppo ja nopea rutiininomainen prosessi. Automaatio suorittaa prosessin aina samalla tavalla versionhallinnan päähaaraan tullessa muutoksia, tätä kutsutaan ohjelmiston julkaisu liukuhihnaksi (engl. Deployment Pipeline). Automaatiolla on myös tarkoitus automatisoida ohjelmiston testaus testiympäristöissä sekä poistaa inhimillisen virheen riski siirroissa eri ympäristöjen välillä. Automaation lopputuloksena tulisikin olla tilanne, jossa ohjelmisto voidaan julkaista milloin vain napin painalluksella tuotantoon. Toinen jatkuvaan toimituksen onnistumiseen tarvittava osa on jatkuva toimitusta tukevan työskentelykulttuurin omaksuminen työyhteisössä. (Fowler, 2013)

Jatkuva toimitusta voidaan vielä jatkaa jatkuvalla julkaisulla, mitä käytettäessä jokainen versionhallinnan päähaaraan tehty muutos julkaistaan automaattisesti tuotantoympäristöön. Jatkuvassa julkaisussa luotetaan siihen, että ohjelmiston testaukseen käytettävät testit kattavat järjestelmän toiminnallisuuden niin hyvin että mahdolliset virheet havaittaisiin niissä. Jatkuvan julkaisun sekä toimituksen suurin ero onkin siinä, että jatkuvaa toimitusta käytettäessä päätöksen siitä onko ohjelmisto valmis julkaistavaksi tuotantoon, tekee henkilö tai henkilöt, jotka vastaavat ohjelmiston julkaisusta toisin kuin jatkuvassa julkaisussa, jossa kaikki julkaisuliukuhihnan läpäisevät versiot julkaistaan automattisesti tuotantoon. (Pittet, d)



Kuva 3. Jatkuvan julkaisun ja toimituksen ero sekä yhteys jatkuvaan integraatioon.

Julkaisuliukuhihna koostuu useista erilaisista vaiheista sekä testausympäristöistä, jotka ovat sitä lähempänä oikeaa tuotantoympäristöä kuin mitä pidemmälle julkaisu on edennyt liukuhihnassa. (Continuous Delivery; Fowler, 2013)

Mukherjeen mukaan julkaisuliukuhihnan neljä yleisintä vaihetta ovat.

- Komponenttivaihe
- Alijärjestelmävaihe
- Järjestelmävaihe
- Tuotantovaihe. (Mukherjee)

Julkaisuliukuhihnan ensimmäisessä vaiheessa eli komponenttivaiheessa käsitellään ohjelmiston pienimpiä jaeltavia sekä testattavia osia. Ensimmäinen vaihe alkaa koodikatselmoinnilla, millä päätetään ovatko koodiin tehdyt muutokset tarpeeksi hyviä, että ne voidaan integroida versionhallinnan päähaaraan. Kun tehdyt muutokset on hyväksytty ohjelma tulisi koostaa automatisoidusti ja ajaa yksikkötestit, joilla voidaan havaita virheet järjestelmän komponenteissa. Testien lisäksi voidaan koodeja tutkia staattisilla analysointityökaluilla, joilla voidaan havaita muistivuotoja sekä samankaltaisuuksia koodissa. (Mukherjee)

Toisessa vaiheessa eli alijärjestelmävaiheessa käsitellään pienimpiä julkaistavia sekä ajettavia yksiköitä. Toisessa vaiheessa järjestelmän osa julkaistaan testiympäristöön, jossa voidaan testata järjestelmän toteuttamiseen tarvittavan osan toimintaa erillään toisista järjestelmän osista. Tässä vaiheessa tulisi järjestelmän osalle ajaa integraatiotestit, joilla testataan järjestelmän osan eri komponenttien yhteistoimintaa. Integraatiotestauksen lisäksi olisi hyvä testata järjestelmän osan suorituskykyä sekä tietoturvaa ajamalla järjestelmän osalle suorituskyky- sekä tietoturvatestit. (Mukherjee)

Liukuhihnan kolmannessa vaiheessa eli järjestelmävaiheessa järjestelmän osat siirretään tuotantoympäristön kaltaiseen testausympäristöön. Jos järjestelmää ei ole kehitetty siten että sen eri osat voitaisiin julkaista erikseen, joudutaan tässä vaiheessa odottamaan muidenkin osien valmistumista, jotta julkaisussa voidaan jatkaa eteenpäin. Tässä vaiheessa tulisi järjestelmälle ajaa myöskin integraatio-, suorituskyky- sekä tietoturvatestit, mutta edellisestä vaiheesta poiketen tässä vaiheessa testit ajetaan tuotantoympäristön kaltaisissa olosuhteissa ja testeillä testataan kaikkien järjestelmän toimintaan vaadittavien osien yhteistoimintaa. Lisäksi järjestelmälle voidaan tehdä myös manuaalista testausta. (Mukherjee)

Neljännessä ja samalla liukuhihnan viimeisessä vaiheessa, eli tuotantovaiheessa, kun järjestelmä on läpäissyt kaikki testit ja saanut tarvittavat hyväksynnät, voidaan vielä päättää, halutaanko muutokset julkaista asiakkaille. Järjestelmän julkaisuun on useita keinoja, esimerkiksi järjestelmä voidaan julkaista vain tietyille asiakkaille tai uusi järjestelmä voi toimia rinnan vanhan kanssa, että sen toiminnasta ollaan täysin varmoja. Järjestelmän julkaisun jälkeen sille ajetaan yleensä vielä savutestit, jotka ovat integraatio- ja tietoturvatestien kaltaisia testejä, joilla pyritään varmistamaan järjestelmän kriittisimpien ominaisuuksien oikeanlainen toiminta. Tämänkin jälkeen järjestelmälle saatetaan vielä tehdä manuaalista testausta. (Mukherjee)

2 YLEISTÄ TYÖKALUISTA

Jatkuvan integraation sekä jatkuvan toimituksen ja julkaisun toteuttamiseen tarvitaan joitain työkaluja. Tässä luvussa käsitellään näitä työkaluja.

2.1 Versionhallintaohjelmistot

Versionhallintajärjestelmä on ohjelmisto, jonka tehtävänä on seurata ohjelmistoprojektin lähdekoodissa tapahtuneita muutoksia. Versionhallinta mahdollistaa useiden kehittäjien yhtaikaisen työskentelyn ohjelmistoprojektissa sekä ohjelmiston eri versioiden vertailun, joka auttaa kehittäjiä löytämään ja korjaamaan muutoksista johtuneet virheet lähdekoodissa. Versionhallinnan ansiosta kehittäjät voivat myös palata edelliseen versioon, jos muutokset ovat rikkoneet jotain mitä ei kyetä korjaamaan tai jos uusi versio on siirretty tuotantoon mutta se ei toimi halutulla tavalla. (Atlassian, a)

Versionhallintaa pidetään yhtenä tärkeimmistä jatkuvan integraation sekä toimituksen mahdollistavista työkaluista. Voidaankin sanoa, että kaikki muutokset, joita ohjelmiston koodeihin tai muihin sen toimintaan vaikuttaviin osiin tehdään, pitäisi tallentaa versionhallintaan. (Rehkopf, b)

Versionhallintaohjelmistoja on kolmenlaisia:

- Paikallinen versionhallinta

  Paikallinen versionhallinta pitää kirjaa tiedostoihin tulleista muutoksista paikallisella koneella. Paikallisessa versionhallinnassa on riskinä, että koneen hajotessa tai tiedostojen korruptoituessa menetetään kaikki, ellei tiedostoja ole varmuuskopioitu.

- Keskitetty versionhallinta

  Keskitetyssä versionhallinnassa kaikki tiedot muutoksista on tallennettu yhdelle keskitetylle palvelimelle, joka mahdollistaa useiden kehittäjien yhtäaikaisen työskentelyn projektin parissa. Keskitetynkin versionhallinnan tapauksessa palvelimen hajotessa tai tiedostojen korruptoituessa on riski menettää kaikki tiedot, ellei varmuuskopioita ole tehty.

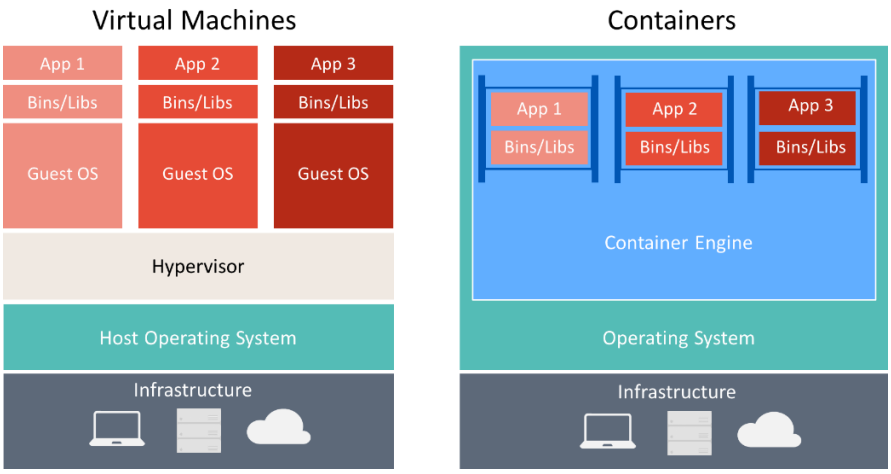
- Hajautettu versionhallinta

  Hajautetussa versionhallinnassa koko historia muutoksista kopioidaan paikalliselle koneelle palvelimelta, joten useiden kehittäjien on mahdollista työskennellä yhtä aikaa projektin parissa kuten keskitetynkin versionhallinnan tapauksessa. Hajautettua versionhallintaa käytettäessä, jos palvelin hajoaa tai korruptoituu, on projektin koko historia edelleen turvassa kehittäjien paikallisilla koneilla, joita voidaan käyttää tietojen palauttamiseen uudelle palvelimelle. (Git, 2014)

2.2 Kontitus

Kontitus on virtualisointitekniikka, jolla pyritään ratkaisemaan erilaisista ympäristöistä johtuvat ongelmat yhdenmukaistamalla ohjelmiston kehitys-, testaus- sekä tuotantoympäristöt. Konttiin on sisällytetty kaikki kontitetun ohjelmiston ajamista varten tarvittavat osat sekä ohjelmistot. Tämä mahdollistaa saman kontitetun ohjelmiston siirtämisen ja ajamisen useissa erilaisissa ympäristöissä, jotka tukevat vain kontin luontiin käytettyä tekniikkaa. (IBM Cloud Education, 2019)

Kontitetun ohjelmiston ajaminen vaatii huomattavasti vähemmän resursseja isäntäjärjestelmältä kuin virtuaalikoneessa ajettava ohjelmisto. Koska jokainen järjestelmässä ajettava kontti jakaa isäntäjärjestelmänsä käyttöjärjestelmän ytimen ja ainoastaan ohjelmiston ajoympäristö virtualisoidaan. Toisin kuin virtuaalikoneissa, joissa virtualisoidaan kaikki aina prosessorista käyttöjärjestelmään asti. Tämän ansiosta palvelimelta vapautuu resursseja ja sillä voidaan ajaa useampaa palvelua. (IBM Cloud Education, 2019)



Kuva 4. Virtuaalikoneiden ja konttien ero.

Konttien käyttäminen parantaa tietoturvaa kontittamattomiin sovelluksiin nähden, koska jokainen kontti ajetaan eristettynä muusta järjestelmästä eivätkä kontit näe muita järjestelmässä pyöriviä ohjelmia tai kontteja. Tämän takia, jos joku onnistuu tunkeutumaan järjestelmään, hän ei näe suoraan kaikkia palvelimella toimivia ohjelmia sekä palveluita. Kontitetusta ympäristöstä on kuitenkin helpompi murtautua ulos kuin virtuaalikoneesta. (IBM Cloud Education, 2019)

Konttien luontiin on useita erilaisia ohjelmistoja, joista eniten käytetty sekä tunnetuin on Docker. Muita kontitus ohjelmistoja on esimerkiksi LXC Linux Containers sekä CoreOS rkt. Konttituksen käytön kasvun myötä kontitustekniikoille on luotu standardeja, joiden käyttöönoton myötä standardien mukaisia kontteja pitäisi pystyä ajamaan kaikilla standardeja tukevilla kontitus ohjelmistoilla. (IBM Cloud Education, 2019)

3 AUTOMATIOPALVELIMEN VALINTAPROSESSI

Tässä luvussa käymme ensimmäisenä läpi erilaisia automatisointipalvelimia sekä palveluita, tämän jälkeen kerrotaan mikä automaatiopalvelin toimeksiantajalle valittiin, sekä mitkä syyt johtivat juuri kyseisen automaatiopalvelimen valintaan.

3.1 Automaatiopalvelimiin tutustuminen

Ohjelmiston koonnin, testauksen sekä julkaisun automatisointiin on saatavilla useita erilaisia automatisointipalvelimia sekä palveluita. Tässä osassa tulemme tutustumaan seuraaviin:

- Bamboo
- Jenkins
- Google Cloud Build
- Bitbucket Pipelines
- Travis CI
- Amazon AWS CodePipeline

Näistä itse ylläpidettäviä automaatiopalvelimia ovat Bamboo, Jenkins, Travis CI ja Bitbucket Pipelines sekä pilvipalveluna saatavia automaatiopalveluita ovat Google Cloud Build, Amazon AWS CodePipeline, Travis CI ja Bitbucket Pipelines.

Seuraavassa kaaviossa on esiteltynä yhteenveto tämän osan teksteistä

	Jenkins	Bamboo	Amazon CodeBuild & CodePipeline	Google Cloud Build	Bitbucket Pipelines	Travis CI
Open Source	Yes	No	No	No	No	No
Hosting type	Saas / On-premises	On-premises	Saas	Saas	Saas / On-premises	Saas / On-premises
Docker support	Via plugins	Support for docker containers	Support for docker containers	Support for docker containers	Support for docker containers	Support for docker containers
Paralel building	Maybe Yes?	Yes	Yes, up to 60 concurrent builds	Yes, up to 10 concurrent builds	Yes, up to 100 concurrent builds	Yes, up to 10 concurrent builds
Test automation	Via plugins	Yes	Yes	Yes	Yes	Yes
Plugins	Yes	Yes	Yes	No	No	No
Supported programming languages	Maybe all?	Any	Any	Any	Any	Over 30 supported programming languages
Supported repositories	Maybe all?	Any	AWS CodeCommit, S3, GitHub and Bitbucket	Google Cloud Storage, Cloud Source Repositories, GitHub and Bitbucket	Only Bitbucket	Only Github
User permission management	Very limited, can be expanded with plugins	Yes	Yes	Yes	Yes	Github user rights management

3.1.1 Jenkins

Jenkins on Javalla kirjoitettu avoimen lähdekoodin ohjelmistotuotannon prosessien automatisointiin tarkoitettu palvelin. Jenkinsillä voidaan automatisoida monenlaisia ohjelmistotuotannon prosesseja ohjelmiston kääntämisestä ja testaamisesta aina ohjelmiston julkaisuun asti. (Jenkins, a)

Jenkins tarjoaa palvelinta ainoastaan itse ylläpidettävänä vaihtoehtona omalle palvelimelle tai pilvipalveluun asennettavaksi. Kuitenkin monet kolmannen osapuolen palveluntarjoajat tarjoavat Jenkinsiä valmiiksi asennettuna vain liukuhihnojen määrittämistä vailla olevana järjestelmänä. (AVI, 2019)

Jenkinsiä pystyy käyttämään Docker-konttien koostamiseen ja liukuhihnan eri vaiheita voidaan ajaa Docker-kontitetuissa ympäristöissä millä voidaan simuloida tuotantoympäristöä. Jenkins kuitenkin tarvitsee kolmannen osapuolen lisäosan tämän toiminnallisuuden toteuttamiseen. (Jenkins, b)

Jenkinsiä käytettäessä on testejä mahdollista ajaa useissa eri liukuhihnan vaiheissa sekä testausympäristöissä. Tämänkin ominaisuuden saadakseen on jälleen kerran käytettävä kolmannen osapuolen lisäosia. (Smartbear)

Jenkinsissä valmiina oleva käyttöoikeuksien hallinta on hyvin rajoittunut. Käyttöoikeuksien parempaan hallintaa on kuitenkin saatavilla lisäosia parantamaan sitä. (Jenkins, c) Lisäosia onkin saatavilla satoja erilaisia lisäosia useisiin erilaisiin tarkoituksiin ohjelmiston koonti, testaus ja tuotantoon siirto prosesseissa sekä moniin muihin tarkoituksiin. (Jenkins, a)

Jenkinsiä koskien ei löydetty luotettavia lähteitä, joista olisi voitu varmistua joistakin tutkimuksessa käytetyistä vertailukohteista. Näistä löytyi kuitenkin keskustelua internetin keskustelupalstoilta, joilta saatujen tietojen mukaan Jenkinsin pitäisi tukea useita yhtaikaisia koonti- ja testausprosesseja. Jenkinsin pitäisi myös tukea kaikkia mahdollisia ohjelmointikieliä sekä versionhallinta ohjelmistoja, mutta näitä käyttääkseen saatetaan joutua asentamaan lisäosia.

3.1.2 Bamboo

Bamboo on Atlassianin suljetun lähdekoodin ohjelmistotuotannon prosessien automatisointiin tarkoitettu palvelin. Bamboolla voidaan automatisoida ohjelmiston koonti, testaus sekä julkaisuprosessit. Bamboo hankitaan yritykselle itse ylläpidettäväksi joko omassa palvelimessa tai kolmannen osapuolen pilvipalvelussa. Bamboon perusversio maksaa 10$ mutta jos tarvitaan mahdollisuus ajaa testausympäristöjä virtuaalikoneissa, hinnoittelu riippuu siitä, kuinka monta virtuaalikonetta tarvitaan. (Atlassian, b)

Kuten Jenkins, Bambookin pystyy koostamaan Docker-kontteja sekä ajamaan liukuhihnan eri vaiheita Jenkinsin tapaan Docker-kontitetuissa ympäristöissä simuloidakseen tuotantoympäristöä, suurin ero onkin, ettei Bamboo tarvitse tähän lisäosia. (Atlassian, 2019a)

Bamboossa pystytään myös ajamaan useita liukuhihnoja yhtäaikaisesti, mutta tällöin voidaan tarvita normaalia parempi lisenssi, jos yhtä aikaa suoritetaan yli 10 tehtävää tai käytetään etänä ajettavia koontiympäristöjä. (Atlassian, b)

Bamboolla voidaan testien ajaminen automatisoida useissa eri liukuhihnan vaiheissa sekä testausympäristöissä. Erityyppisiä testejä voidaan ajaa myös rinnakkain testiprosessin aikana. (Atlassian, b)

Bamboollekin on saatavilla kolmannen osapuolen lisäosia muun muassa muilta Bamboon käyttäjiltä sekä Atlassian Marketplacen kumppaneilta, jos niille ilmenee tarvetta. (Atlassian, 2019d)

Bamboolla voidaan koota millä tahansa ohjelmointikielellä ohjelmoitu sovellus (Atlassian, c), minkä lähdekoodit voidaan hakea minkälaisesta versionhallintaohjelmistosta tahansa (Atlassian, 2019c).

Bamboossa itsessään on valmiiksi käyttäjienhallinta millä voidaan rajoittaa käyttäjien oikeutta käyttää mitäkin Bamboon ominaisuutta. (Atlassian, 2019e)

3.1.3 Google Cloud Build

Google Cloud Build on googlen tuottama suljetun lähdekoodin jatkuvan integraation sekä julkaisun pilvipalvelu. Google Cloud Buildilla voidaan automatisoida useimmat ohjelmiston koontiin, testaukseen sekä julkaisuun liittyvien prosessien vaiheet. Google Cloud Buildiin kuuluu 120 minuuttia ilmaista käännös aikaa päivässä minkä jälkeen lisäminuutit ovat 0.003$ minuutilta. (Google, a)

Google Cloud Build tukee Dockeria suoraan ja sitä voidaankin käyttää Docker-kontin tekemiseen ohjelmistosta. Google Cloud Buildissa jokainen julkaisuliukuhihnan eri vaihe ajetaan Docker-kontitetetussa ympäristössä. (Google, b)

Google Cloud Buildissa voidaan normaalisti ajaa yhtaikaa kymmentä erillistä liukuhihnaa. Määrän ylittyessä uudet koonti- ja testausprosessit jätetään odottamaan vanhojen valmistumista. Vuoroaan odottamassa olevien koonti- ja testausprosessien määrää ei ole rajoitettu mitenkään. (Google, 2019a)

Google Cloud Buildia käytettäessä voidaan testien ajaminen automatisoida useissa eri liukuhihnan vaiheissa sekä testausympäristöissä. Erityyppisiä testejä voidaan aja myös rinnakkain testiprosessin aikana (Google, a). Tämän lisäksi Google Cloud Buildissa voidaan rajoittaa millä käyttäjäryhmillä on oikeus käyttää mitäkin Cloud Buildin ominaisuutta. (Google, 2019b)

Google Cloud Build voi hakea projektin lähdekoodit ainoastaan Google Cloud Storagesta, Cloud Source Repositoriesista, GitHubista tai Bitbucketista. Cloud Buildissa kaikki ohjelmointikielet ovat tuettuja koska ohjelmat käännetään ja testataan Docker-kontitetuissa ympäristöissä. (Google, b)

3.1.4 Amazon AWS CodePipeline & CodeBuild

Amazon AWS CodePipeline on Amazonin tuottama suljetun lähdekoodin jatkuvan integraation sekä julkaisun pilvipalvelu, jolla voidaan automatisoida useimmat ohjelmiston koontiin, testaukseen sekä

julkaisuun liittyvä liukuhihnat (Amazon, d). CodePipelinen käyttö maksaa 1$ kuussa per aktiivinen liukuhihna. (Amazon, g)

Amazon AWS CodePipelinessa ei itsessään pysty automatisoimaan ohjelman kääntämistä. Sen sijaan tähän on käytettävä toista Amazonin tarjoamaa ohjelmistoa Amazon AWS CodeBuildia (Amazon, a). CodeBuildin käytöstä maksetaan käytettyjen käännös minuuttien mukaan. Yhden kääntämiseen käytetyn minuutin hinta määräytyy sen mukaan mitä enemmän resursseja kääntämiselle halutaan. Kääntäminen maksaa halvimmillaan vähimmäismäärällä resursseja 0.005$ minuutilta (Amazon, h). Amazon AWS CodeBuildissa on mahdollista ajaa yhtaikaa enintään 60 koonti- sekä testausprosessia. (Amazon, b)

Amazon AWS CodeBuildilla voidaan automatisoida ohjelmiston koonti- sekä testausprosessit, sekä koska CodeBuild tukee Dockeria suoraan, sillä voidaan ohjelmistoista koostaa Docker-kontteja julkaisuvalmiiksi. (Amazon, c)

Amazon AWS CodePipelineen on mahdollista yhdistää itse tehtyjä sekä kolmannen osapuolen tekemiä lisäosia auttamaan liukuhihnojen automatisoinnissa (Amazon, a). Minkä lisäksi Amazon AWS CodePipelinen käyttäjien hallintaan voidaan käyttää Amazonin käyttäjien hallintaa, jolla voidaan rajoittaa mitä CodePipelinen ominaisuuksia käyttäjä voi käyttää. (Amazon, f)

Amazon AWS CodePipeline voi hakea projektin lähdekoodit ainoastaan AWS CodeCommitista, S3sta, GitHubista ja Bitbucketista. CodePipeline kaikki ohjelmointikielet ovat tuettuja koska ohjelmat käännetään sekä testataan Docker-kontitetuissa ympäristöissä. (Amazon, e)

3.1.5 Bitbucket Pipelines

Bitbucket Pipelines on Atlassianin Bitbucket versionhallintatyökalussa mukana oleva suljetun lähdekoodin jatkuvan integraation sekä julkaisun työkalu. Bitbucket Pipelinesillä voidaan automatisoida useimpia ohjelmiston koontiin, testaukseen sekä julkaisuun liittyvien prosessien vaiheita. Bitbucket on saatavilla pilvipalveluna tai omalla palvelimella itseylläpidettävänä ohjelmistona. Jokaiseen pilvipalveluna ostettuun tilaukseen kuuluu vähintään 50 minuuttia ilmaista käännösaikaa kuukaudessa. Lisäminuutit maksavat 10$ per 1000 minuuttia. (BitBucket, a)

Bitbucket Pipelines tukee Dockeria suoraan ja sitä voidaan käyttää Docker kontin koostamiseen ohjelmistosta. Bitbucket Pipelinessä liukuhihnan eri vaiheet ajetaan Docker-kontitetuissa ympäristöissä millä voidaan simuloida erilaisia testaus- ja tuotantoympäristöjä. (Bitbucket, b)

Bitbucket Pipelinessä on mahdollista ajaa useaa yhtäaikaista liukuhihnaa mutta yhtaikaa ajettavien liukuhihnojen määrä riippuu käytettävästä Bitbucketin versiosta. Ilmaisessa versiossa yhtaikaisen liukuhihnojen määrä on rajattu kymmeneen, ja maksullisissa versioissa sataan yhtaikaiseen liukuhihnaan. (Bitbucket, c)

Bitbucket Pipelinesillä voidaan testien ajaminen automatisoida useissa eri liukuhihnan vaiheissa sekä testausympäristöissä. Erityyppisiä testejä voidaan aja myös rinnakkain testiprosessin aikana. (BitBucket, a)

Bitbucket Pipeline tukee kaikkia ohjelmointikieliä, jotka toimivat Linuxilla, koska ohjelmat käännetään ja testataan Docker-kontitetuissa ympäristöissä (Atlassian, 2019b). Minkä lisäksi Bitbucket Pipelinessä voidaan rajoittaa ketkä käyttäjistä saavat julkaista sovelluksen tuotantoon sekä mistä versionhallinnan haaroista sovellus voidaan julkaista. (Hodder, 2019)

3.1.6 Travis CI

Travis on suljetun lähdekoodin jatkuvan integraation sekä julkaisun automatisointipalvelin, jolla voidaan automatisoida useimpia ohjelmiston koontiin, testaukseen sekä julkaisuun liittyvien prosessien vaiheita. (Travis CI, a) Travis on saatavilla pilvipalveluna tai omalla palvelimella itse ylläpidettävänä ohjelmistona. Traviksen halvin tilaus maksaa 63$ kuussa, mihin kuuluu loputtomasti käännösaikaa, mutta vain yksi yhtaikaa ajettava liukuhihna. Jos yhtaikaisia liukuhihnoja tarvitsee enemmän, on vaihdettava parempaan tilukseen Travisista. (Travis Ci, b)

Travisia pystyy käyttämään Docker-konttien koostamiseen, sekä liukuhihnan eri vaiheita voidaan ajaa Docker-kontitetuissa ympäristöissä millä voidaan simuloida erilaisia tuotantoympäristöjä (Travis CI, c). Kuitenkin Traviksessa käyttäjien hallinta perustuu GitHubin käyttäjien hallintaan, joten kaikki, joilla on oikeus tehdä muutoksia voivat käynnistää liukuhihnan. (Travis CI, d)

Travisia käytettäessä voidaan testien ajaminen automatisoida useissa eri liukuhihnan vaiheissa sekä testausympäristöissä. Travis tukee yli 30 eri ohjelmointikieltä, mutta projektin lähdekoodien versionhallintaohjelmistona tuetaan ainoastaan GitHubia. (Travis CI, a)

3.1.7 Muut

Aikaisemmin esiteltyjen automaatiopalvelinten sekä -palveluiden lisäksi on olemassa muitakin, joita ei kuitenkaan otettu vertailuun mukaan. näitä ovat esimerkkisi: Azure DevOps, Circle CI, TeamCity, GitLab CI/CD, Buddy, CodeShip, GoCD, Wercker, Semaphore, Nevercode, Spinaker ja BuildBot. Näiden pois jäänneille on useita syitä, kuten esimerkiksi osa on tarkoitettu ainoastaan julkaisun automatisointiin, osa on kehitetty pääasiassa mobiiliohjelmistojen kehityksen automatisointia varten ja varmaankin suurimpana syynä ettei työn määrä kaikkia eri vaihtoehtoja tutkiessa paisuisi valtavaksi.

3.2 Automaatiopalvelimen valinta

Automaatiopalvelimeksi toimeksiantajalle valittiin Jenkins. Jenkinsiin valinnalle oli useita syitä kuten se että se on maksuton, sekä koska se ei sitouttaisi toimeksiantajaa minkään palveluntarjoajan järjestelmään. Tämän takia sillä olisi hyvä testata ja hankkia kokemusta jatkuvan integraation sekä

jatkuvan julkaisun käyttöön liittyvistä toimintatavoista sekä ohjelmiston kääntämisen, testaamisen ja tuotantoon siirtämisen automatisoinnista. Muita syitä Jenkinsin valintaan oli muun muassa se, että sen logo oli joidenkin valintaan osallistuneiden henkilöiden mielestä parhaan näköinen.

4 TYÖN TOTEUTUS

Tässä luvussa käydään läpi Jenkinsin asennusta sekä sen asetusten määritysprosessia sekä esitellään Jenkinsillä toteutettua automatisoitua jatkuvan integraation sekä jatkuvan julkaisun liukuhihnaa.

4.1 Jenkinsin asennusprosessi

Jenkins asennettiin sitä varten pystytetylle palvelimelle, minkä käyttöjärjestelmänä oli 64-bittinen CentOS 7.7 Linux-jakelu. Palvelimen pystytys itsessään oli automatisoitu Ansiblella, joten sille oli asennettu valmiiksi kaikki toimeksiantajan palvelimillaan vaatimat ohjelmistot, sekä näiden asetukset olivat määritetty toimeksiantajan vaatimusten mukaisiksi. Koska Jenkinsin ohjauspaneelia käytettiin selainpohjaisesti, palvelimelle reititettiin oma nettiosoite. Jenkinsin asentamisessa käytettiin apuna Jenkinsin Wikissä löytynyttä asennusohjetta sekä useita muita netistä löytyviä Jenkinsin asentamiseen liittyviä dokumentteja.

Jenkinsin asennusta varten oli palvelimelle ensiksi asennettava OpenJDK, joka on avoimen lähdekoodin implementointi Oraclen Java standard editionista. Koska Jenkins ei ollut kyseisen Linux-jakelun paketinhallinnan peruspaketti, oli lähde Jenkinsin paketille lisättävä palvelimen paketinhallintaan, jotta se voitaisiin asentaa sekä päivittää paketinhallinnan kautta.

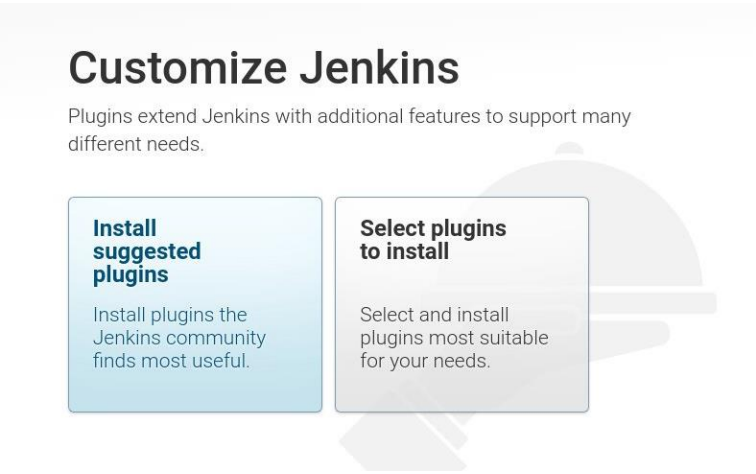
Jotta tietoliikenne Jenkinsin ohjauspaneeliin saatiin suojattua, tarvittiin nettiosoitteelle oma TLS/SSL-sertifikaatti. Sertifikaatti tehtiin käyttämällä ilmaista Let's Encrypt-nimistä palvelua, joka on voittoa tavoittelemattoman järjestön Internet Security Research Groupin (ISRG) tarjoama palvelu. Palvelun käyttämistä varten oli palvelimelle asennettava Electronic Frontier Foundationin (EFF) tuottama Certbot-niminen ohjelma, joka tarkistettuaan, että juuri kyseinen palvelin omistaa kyseisen nettiosoitteen pyytää Let's Encrypt palvelua myöntämään sertifikaatin kyseiselle nettiosoitteelle.

Jotta Jenkinsin ohjauspaneeliin saataisiin suojattu HTTPS-yhteydellä, piti sitä varten hankittu sertifikaatti ottaa käyttöön. Kuitenkin jotta olisi voitu käyttää Jenkinsin sisäänrakennettua HTTPS-palvelua, olisi sertifikaatista pitänyt tehdä Java KeyStore (JKS) mikä osoittautui todella suureksi ja monimutkaiseksi operaatioksi. Tämän takia ja koska Jenkinsin sertifikaatin asennusohjeessa näin suositeltiin, päädyttiin serverille asentamaan NGINX web-palvelin. NGINXin asentamisen jälkeen käytettiin jälleen Certbotia, jolla pystyttiin lisäämään automaattisesti Let's Encrypt palvelun myöntämä sertifikaatti NGINX-palvelimelle. Tämän jälkeen asetettiin NGINX reitittämään palvelimen nettiosoitteeseen tuleva liikenne palomuurin takana sijaitsevalle Jenkinsille.

Tietoliikennettä varten oli Jenkins palvelimen palomuuriin avattava tarvittavat portit, jotta yhteys NGINX web-palvelimelle olisi mahdollista saada. Yhteys NGINX web-palvelimelle tämän jälkeen toimikin mutta reititys Jenkinsille ei. Virhe raportteja tutkimalla selvisi lopulta, että NGINXin verkko osoitteeseen tulevan liikenteen reititys Jenkinsille esti toimeksiantajan käyttämä käyttöoikeuksien hallinta ohjelmisto SELinux. Kun SELinux oli asetettu sallimaan liikenteen reititys NGINXiltä Jenkinsille päästiin vihdoinkin viimeistelemään Jenkinsin asennusta.

Kuva 5. Kuva Jenkinsin aloitussalasanan tarkistus dialogista.

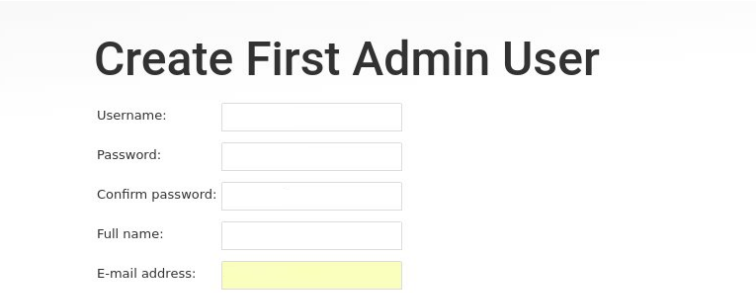
Jenkinsin asennus viimeisteltiin menemällä Jenkinsin ohjauspaneelin nettiosoitteeseen missä aukesi kuvan 5 mukainen dialogi. Dialogissa kysyttiin Jenkinsin pääkäyttäjän tunnuksen automaattisesti luotua oletussalasanaa. Salasana saatiin käyttämällä cat-komentoa dialogin ilmoittamassa kansiossa olevaan initialAdminPassword-tiedostoon.



Kuva 6. Kuva Jenkinsin kustomointi dialogista.

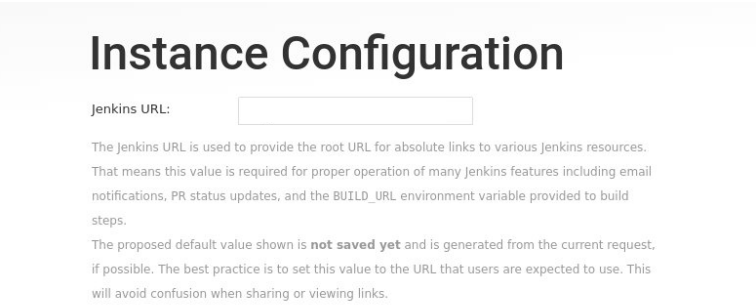
Sisäänkirjautumisen jälkeen päästiin kuvan 6 näköiseen dialogiin, jossa kysyttiin, halutaanko käyttää ehdotettuja lisäosia vai valita itse asennettavat lisäosat. Valitsemalla haluavamme valita itse asennettavat lisäosat päästiin lisäosien valinta dialogiin, josta pystyttiin valitsemaan itse mitkä

lisäosat asennetaan Jenkinsin asennuksen viimeistelyn yhteydessä. Kun asennettavat lisäosat olivat valittu, päästiin dialogiin, jossa esitettiin lisäosien asennuksen edistyminen.



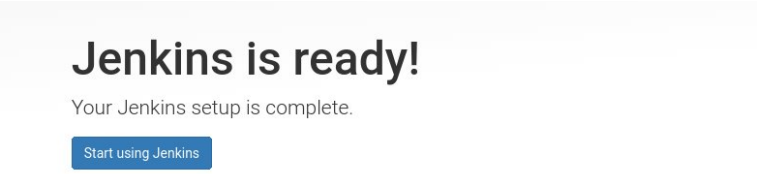
Kuva 7. Kuva Jenkinsin ensimmäisen Admin käyttäjän luomisdialogista.

Tämän jälkeen päästiin kuvan 7 mukaiseen dialogiin, jossa tuli luoda ensimmäinen Admin käyttäjä Jenkinsiin, millä Jenkinsin asetukset tultaisiin määrittämään.



Kuva 8. Kuva Jenkinsin nettiosoitteen määritys dialogista.

Käyttäjän luonnin jälkeen päästiin kuvan 8 mukaiseen dialogiin, jossa tuli asettaa Jenkinsille tieto nettiosoitteesta, josta siihen pääsee yhdistämään internetistä. Jenkins tarvitsee tämän tiedon siksi että jos sen tarvitsee luoda linkkejä joihinkin sen lähettämiin viesteihin se tietäisi mistä nettiosoitteesta siihen pääse kiinni.



Kuva 9. Kuva Jenkinsin asennuksen viimeisestä dialogista.

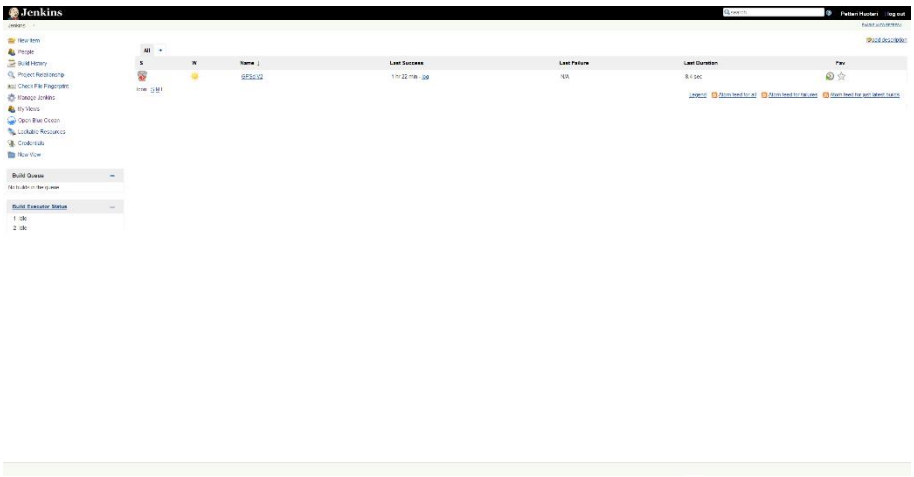
Nettiosoitteen asettamisen jälkeen päästiin kuvan 9 mukaiseen dialogiin, jossa ilmoitettiin Jenkinsin asennuksen olevan valmis. Start using Jenkins-napista painamisen jälkeen päästiin Jenkinsin sisäänkirjautumis- sivulle ja voitiin kirjautua asennuksen aikana luoduilla tunnuksilla sisään Jenkinsiin.

Alla joitain Jenkinsin asennukseen tarvittuja komentorivi komentoja.

```
# sudo yum install java-11-openjdk
# sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
# sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
# sudo yum install jenkins
# sudo systemctl start jenkins
# sudo systemctl enable jenkins
# sudo yum install epel-release
# sudo yum install nginx
# sudo systemctl enable nginx
# sudo systemctl start nginx
# sudo yum install certbot certbot-nginx
# sudo certbot certonly –nginx
# sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

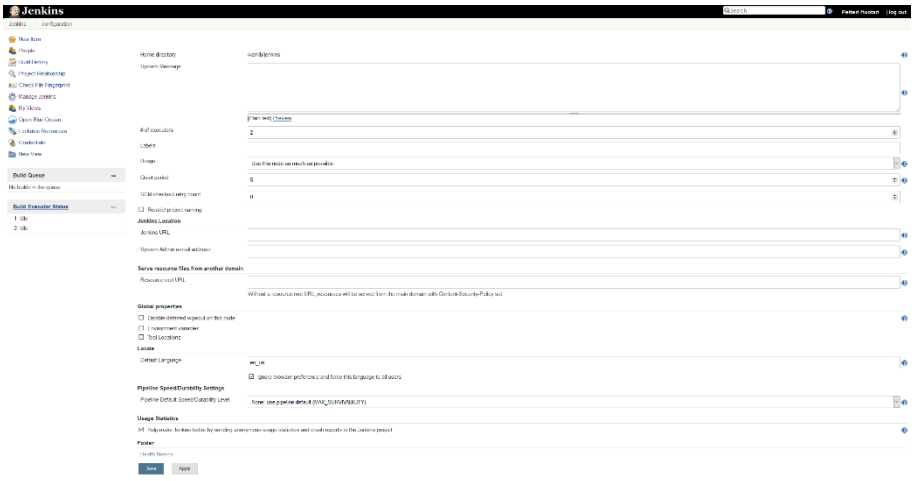
4.2 Jenkinsin asetusten määrittäminen

Kun Jenkins oli saatu asennettua tuli aika asentaa tarvittavat lisäosat, jotka olivat jääneet asentamatta asennusprosessin yhteydessä, sekä määrittää Jenkinsin sekä siihen asennettujen lisäosien yleiset asetukset. Jenkinsin sekä siihen asennettujen lisäosien yleisiä asetuksia voidaan märittää käyttämällä Jenkinsin selainpohjaista hallintapaneelia, mistä voidaan myös asentaa uusia lisäosia tarvittaessa. Jenkinsin yleisten asetusten määrittämisen yhteydessä määritettiin myös tarvittavat SSH avaimet sekä muut tarvittavat käyttäjä tunnukset.



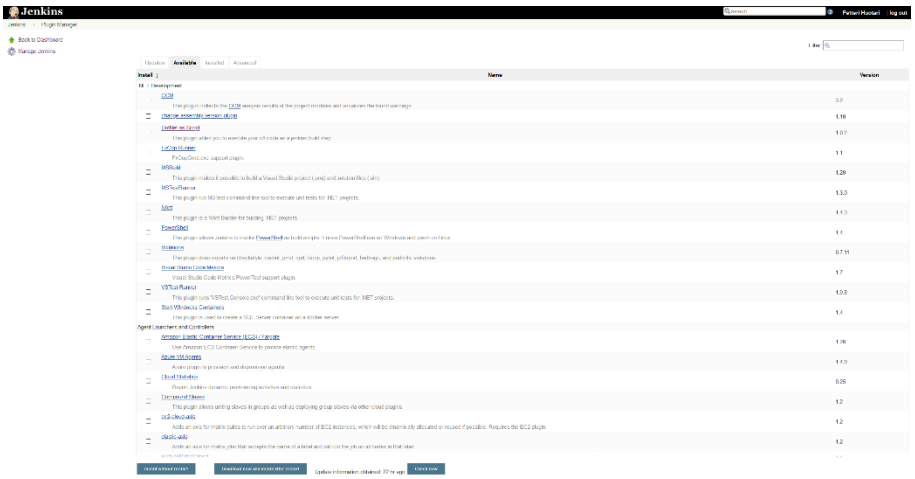
Kuva 10. kuva Jenkinsin hallintapaneelin pääsivusta.

Ensimmäisellä kirjautumisella päästiin suunnilleen kuvan 10 näköiselle Jenkinsin hallintapanelin etusivulle. Jenkinsin hallintapanelin etusivulla on vasemmassa laidassa aivan ylimpänä valikko, josta pääsee melkein kaikkiin Jenkinsin hallintaan käytettäviin valikoihin. Jotkin valikoista on kuitenkin piilotettu ja niihin päästäkseen on tiedettävä URL-osoite mistä ne löytyvät. Valikon alapuolella on lista, josta näkyy tällä hetkellä suoritettavana olevat käännös- ja testausprosessit. Listan alapuolella on toinen lista, jossa näkyy kaikkien käännös- ja testausprosesseihin käytettävien prosessien tila.



Kuva 11. Kuva Jenkinsin Configure System valikosta.

Suurinta osaa Jenkinsin sekä sen lisäosien yleisistä asetuksista voidaan hallita kuvassa 11 näkyvästä Configure System valikosta, mikä teki siitä varmaan eniten työn aikana käytetyn asetusvalikon. Suurinta osaa alussa tehdyistä asetusmäärityksistä jouduttiin kuitenkin muokkaamaan työn aikana sitä mukaa kun liukuhihna valmistui.



Kuva 12. Kuva Jenkinsin Manage Plugins Valikosta.

Jenkinsiin voitiin asentaa uusia lisäosia sekä päivittää jo asennettuja lisäosia käyttäen kuvassa 12 näkyvää Manage Plugins-valikkoa. Jenkins tarvitsee useiden ominaisuuksien toteuttamiseen kolmannen osapuolen lisäosia, joita jouduttiinkin asentamaan melko runsaasti.

Yleisten asetusten määrittämisen lisäksi tuli Jenkinsiin ohjelmoida koonti, testaus ja julkaisu liukuhihna projektille. Liukuhihna ohjelmoitiin esimerkkinä toimivan sovelluksen versionhallinnassa olevaan Jenkinsfile-nimiseen tiedostoon mistä Jenkins lukee sen. Tämä mahdollistaa sen, että tarvittaessa kuka tahansa kehittäjätiimistä voi muokata liukuhihnaa, mikäli siinä ilmenee puutteita tai virheitä.

4.3 Esimerkit käyttötapauksista

Ensimmäiseksi sekä tässä opinnäytetyössä raportoitavaksi jatkuvan- integraation ja julkaisun liukuhihna toteutettiin toimeksiantajan GPSd v2 nimiselle GPS-datan vastaanotto palvelimelle. GPSd v2:n tapauksessa liukuhihna koostui viidestä vaiheesta.

Nämä viisi vaihetta ovat:

- Alustusvaihe

  Liukuhihnan ensimmäisessä vaiheessa Jenkins testaa, että kääntämiseen ja testaamiseen tarvittavat työkalut ovat asennettuina. Työkalujen testaamisen jälkeen Jenkins hakee muuttuneen haaran versionhallinnasta. Jenkins myös mahdollisesti yhdistää versionhallinasta haetun haaran toiseen, mikäli kyse on yhdistämispyynnön takia käynnistetystä liukuhihnasta.

- Käännösvaihe

  Liukuhihnan toisessa vaiheessa Jenkins kääntää GPSd v2:n versionhallinnasta haetusta lähdekoodista. Mikäli kääntäminen epäonnistuu Jenkins siirtyy suoraan liukuhihnan viimeisenä olevaan viimeistelyvaiheeseen.

- Testausvaihe

  Liukuhihnan kolmannessa vaiheessa Jenkins ajaa käännetylle GPSd v2:lle kaikki sen testiprojektissa olevat testit. Testien ajamisen jälkeen Jenkins tekee raportin testituloksista mikä kehittäjien on mahdollista nähdä Jenkinsin ajon tuloksista. Mikäli testauksessa tulee virheitä, siirrytään suoraan viimeistelyvaiheeseen.

- Julkaisuvaihe

  Liukuhihnan neljännessä vaiheessa ensimmäisenä Jenkins lähettää viestin julkaisun aloittamisesta julkiselle Slack-kanavalle, mikäli viestin lähettäminen epäonnistuu, jätetään julkaisu tekemättä. Viestin lähettämisen jälkeen Jenkins siirtää käännetyn GPSd v2:n tuotantopalvelimelle missä ajetaan julkaisuun käytettävä skripti, joka sammuttaa vanhan palvelimen, kopioi tiedostot oikeaan kansioon ja käynnistää uuden palvelimen.

- Viimeistelyvaihe

  Liukuhihnan viimeisessä vaiheessa Jenkins raportoi kääntämisen sekä testaamisen tuloksen kehittäjälle Slack-viestillä. Mikäli kehittäjän Slack-tunnusta ei voitu jostain syystä selvittää lähetetään viesti julkiselle Slackissä olevalle raportointi kanavalle. Liukuhihna lähettää myös versionhallinnalle tiedon muutoksen statuksesta, jotta versionhallinta ohjelmisto voi merkata sen muutokselle.

GPSd v2:n tapauksessa erilaisia liukuhihnan käyttötapauksia oli työn tekemisen hetkellä kolme kappaletta.

Nämä kolme käyttötapausta ovat:

- Versionhallinnan haaraan tulleen muutoksen koonti ja testaus.

  Tässä tapauksessa kehittäjä tekee johonkin versionhallinnassa olevaan haaraan muutoksen. Mikä laukaisee Jenkins-liukuhihnan, josta suoritetaan kaikki muut vaiheet paitsi julkaisuvaihe.

Kuva 13. Liukuhihnan vaiheet

- Versionhallinnan yhdistämispyyntöön tulleen tai vaikuttavan muutoksen koonti ja testaus.

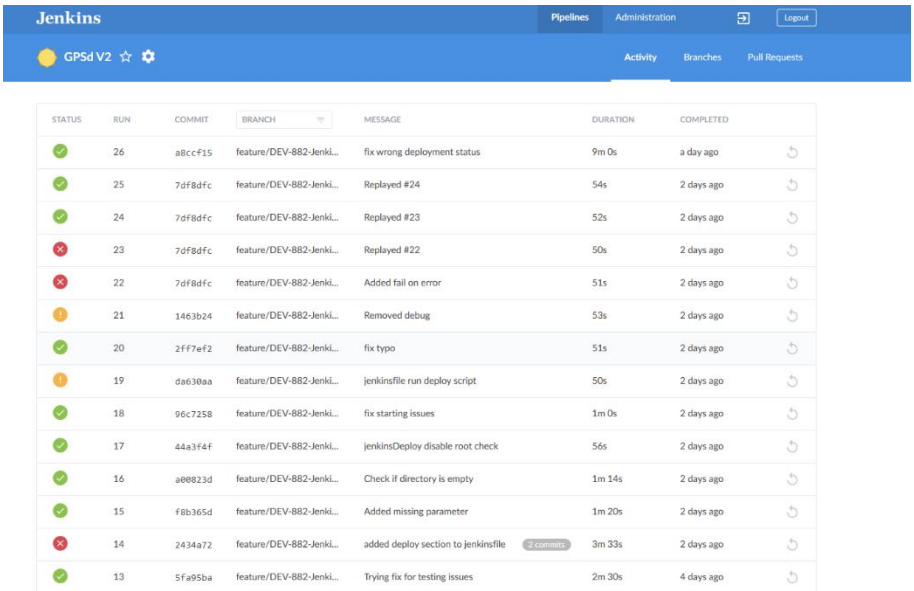
  Tässä tapauksessa kehittäjä tekee johonkin versionhallinnassa olevaan haaraan muutoksen, josta on tehty yhdistämispyyntö, tai haaraan, johon on tehty yhdistämispyyntö. Minkä laukaisemassa Jenkins-liukuhihnassa Jenkins ensiksi automaattisesti yhdistää paikallisesti versionhallinnan haarat toisiinsa ja mikäli virheitä ei synny suorittaa liukuhihnasata kaikki muut vaiheet paitsi julkaisuvaiheen.

- Versionhallinnan testaus haaraan tulleen muutoksen koonti, testaus sekä julkaisu

  Tässä tapauksessa kehittäjä tekee versionhallinnassa olevaan testaushaaraan muutoksen tai siihen tehty yhdistämispyyntö hyväksytään. Mikä laukaisee Jenkins-liukuhihnan, josta suoritetaan kaikki vaiheet.

Kuva 14. Liukuhihnan vaiheet julkaisun kanssa.

Kehittäjät voivat käydä lukemassa tarkemmat tiedot jokaisen muutoksen käännös- sekä testausprosessien tuloksista Jenkinsin hallintapanelin yhteydessä olevalla raportointi sivulta.



Kuva 15. Raportti eri muutosten kääntämisen sekä testaamisen lopputuloksista.

Tulevaisuudessa on tarkoitus laittaa kuvassa 14 näkyvä infoikkuna näkymään toimistolla olevalle infonäytölle. Sekä myös tuotantoon julkaisukin on tarkoituksena automatisoida, kunhan liukuhihna on ollut jonkin aikaa testattavana versionhallinnan testihaaran julkaisussa, jotta on varmistuttu, että liukuhihna toimii varmasti halutulla tavalla. Tarkoituksena on myös tehdä omat liukuhihnat myös muille toimeksiantajan ohjelmistoprojekteille.

5 YHTEENVETO

Tämän opinnäytetyön aiheena oli Jatkuvan integraation sekä julkaisun järjestelmän lisääminen osaksi toimeksiantajan ohjelmistokehitys prosessia. Järjestelmän käyttöönoton tarkoituksena oli helpottaa sekä nopeuttaa muutosten tekemistä, testausta ja muutosten julkaisua asiakkaille.

Omasta ja toimeksiantajan mielestä opinnäytetyön lopputuloksena saatu järjestelmä täyttää toimeksiantajan sille alussa asettamat vaatimukset, vaikkakin alun perin opinnäytetyön yhteydessä toteutettavaksi suunniteltu Docker kontitus päätettiin jättää odottamaan, kunnes sitä tukemaan tarvittava infrastruktuuri mahdollistaa siirtymisen käyttämään sitä.

Mielestäni Jenkinsin käyttö automaatiopalvelimena oli oikein hyvä valinta, koska melkein kaikkeen oli saatavilla lisäosa ja jos ei ollut oli toiminnallisuuden ohjelmoiminen itse melko helppoa. Jenkinsillä on myös laaja käyttäjä yhteisö, joten internetistä löytyi helposti paljon keskustelua ja esimerkkejä erilaisista toteutuksista.

Opinnäytetyön tekemisen aikana ei kohdattu mitään suurempia ongelmia vaikkakin SELinux ei aluksi tahtonut suostua yhteistoimintaan. Ongelmana oli myös Javan KeyStore, minkä takia jouduttiin asentamaan NGINX web-palvelin sertifikaatteja varten. Myöhemmin NGINXin asentaminen tosin osoittautui erittäin hyväksi asiaksi koska se mahdollisti liikenteen estämisen muista kuin toimiston ja sisäverkon osoitteista helposti. Tämän lisäksi Jenkinsiin saatavilla ollut Slack-lisäosa ei tukenut toimeksiantajan haluaman mukaista viestien lähettämistä, joten niiden lähettämisen toiminnallisuus oli ohjelmoitava itse.

Jatkokehityksenä järjestelmän toiminnallisuuksia olisi hyvä pilkkoa pienempiin eri projektien kesken jaettuihin osiin parantamaan ylläpidettävyyttä. Myös tulevaisuudessa Jenkins palvelimen pystytysprosessi olisi hyvä automatisoida täysin Ansiblella. Tämä mahdollistaisi tilanteessa, jossa palvelin menee rikki Jenkinsin nopean palauttamisen käyttöön

LÄHTEET

Amazon. (a). *aws.amazon.com*. Haettu 13. 1. 2020 osoitteesta https://aws.amazon.com/codepipeline/features/

Amazon. (b). *docs.aws.amazon.com*. Haettu 13. 1. 2020 osoitteesta https://docs.aws.amazon.com/codebuild/latest/userguide/limits.html

Amazon. (c). *aws.amazon.com*. Haettu 13. 1. 2020 osoitteesta https://aws.amazon.com/codebuild/features/?nc=sn&loc=2

Amazon. (d). *aws.amazon.com*. Haettu 7. 1. 2020 osoitteesta https://aws.amazon.com/codepipeline/

Amazon. (e). *aws.amazon.com*. Haettu 16. 1. 2020 osoitteesta https://aws.amazon.com/codebuild/faqs/

Amazon. (f). *docs.aws.amazon.com*. Haettu 17. 1. 2020 osoitteesta https://docs.aws.amazon.com/codepipeline/latest/userguide/security-iam.html

Amazon. (g). *aws.amazon.com*. Haettu 20. 1. 2020 osoitteesta https://aws.amazon.com/codepipeline/pricing/

Amazon. (h). *aws.amazon.com*. Haettu 20. 1. 2020 osoitteesta https://aws.amazon.com/codebuild/pricing/

Atlassian. (23. 9. 2019a). *confluence.atlassian.com*. Haettu 10. 1. 2020 osoitteesta https://confluence.atlassian.com/bamboo/getting-started-with-docker-and-bamboo-687213473.html

Atlassian. (28. 2. 2019b). *confluence.atlassian.com*. Haettu 16. 1. 2020 osoitteesta https://confluence.atlassian.com/bitbucket/get-started-with-bitbucket-pipelines-792298921.html

Atlassian. (17. 9. 2019c). *confluence.atlassian.com*. Haettu 16. 1. 2020 osoitteesta https://confluence.atlassian.com/bamboo/linking-to-source-code-repositories-671089223.html

Atlassian. (17. 4. 2019d). *developer.atlassian.com*. Haettu 10. 1. 2020 osoitteesta https://developer.atlassian.com/server/bamboo/

Atlassian. (9. 8. 2019e). *confluence.atlassian.com*. Haettu 17. 1. 2020 osoitteesta https://confluence.atlassian.com/bamboo/bamboo-permissions-369296034.html

Atlassian. (a). *atlassian.com*. Haettu 11. 1. 2020 osoitteesta https://www.atlassian.com/git/tutorials/what-is-version-control

Atlassian. (b). *atlassian.com*. Haettu 7. 1. 2020 osoitteesta https://www.atlassian.com/software/bamboo

Atlassian. (c). *atlassian.com*. Haettu 16. 1. 2020 osoitteesta https://www.atlassian.com/software/bamboo/features

AVI. (24. 9. 2019). *geekflare.com*. Haettu 10. 1. 2020 osoitteesta https://geekflare.com/jenkins-hosting-platform/

BitBucket. (a). *bitbucket.org*. Haettu 3. 1. 2020 osoitteesta https://bitbucket.org/product/features/pipelines

Bitbucket. (b). *confluence.atlassian.com*. Haettu 11. 1. 2020 osoitteesta https://confluence.atlassian.com/bitbucket/run-docker-commands-in-bitbucket-pipelines-879254331.html 11.1

Bitbucket. (c). *confluence.atlassian.com*. Haettu 11. 1. 2020 osoitteesta https://confluence.atlassian.com/bitbucket/limitations-of-bitbucket-pipelines-827106051.html

Cohn, M. (17. 12. 2009). *MountaingoatSoftware.com*. Haettu 8. 1. 2020 osoitteesta https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid 8.1

Continuous Delivery. (ei pvm). *continuousdelivery.com*. Haettu 30. 12. 2019 osoitteesta https://continuousdelivery.com/foundations/test-automation/

Fowler, M. (1. 5. 2006). *martinFowler.com*. Haettu 19. 12. 2019 osoitteesta https://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration

Fowler, M. (30. 5. 2013). *martinFowler.com*. Haettu 23. 12. 2019 osoitteesta https://martinfowler.com/bliki/ContinuousDelivery.html

Git. (2014). *git-scm.com*, 2nd Edition. Haettu 2. 1. 2020 osoitteesta https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Google. (3. 12. 2019a). *cloud.google.com*. Haettu 10. 1. 2020 osoitteesta https://cloud.google.com/cloud-build/quotas

Google. (16. 12. 2019b). *cloud.google.com*. Haettu 17. 1. 2020 osoitteesta https://cloud.google.com/cloud-build/docs/securing-builds/configure-access-control

Google. (a). *cloud.google.com*. Haettu 7. 1. 2020 osoitteesta https://cloud.google.com/cloud-build/

Google. (b). *cloud.google.com*. Haettu 7. 1. 2020 osoitteesta https://cloud.google.com/cloud-build/docs/

Hodder, P. (8. 4. 2019). *bitbucket.org*. Haettu 17. 1. 2020 osoitteesta https://bitbucket.org/blog/deployment-permissions-now-available-in-bitbucket-pipelines

IBM Cloud Education. (15. 5. 2019). *ibm.com*. Haettu 31. 12. 2019 osoitteesta https://www.ibm.com/cloud/learn/containerization

Jenkins. (a). *jenkins.io*. Haettu 10. 1. 2020 osoitteesta https://jenkins.io/

Jenkins. (b). *jenkins.io*. Haettu 10. 1. 2020 osoitteesta https://jenkins.io/doc/book/pipeline/docker/

Jenkins. (c). *jenkins.io*. Haettu 11. 3. 2020 osoitteesta https://wiki.jenkins.io/display/JENKINS/Standard+Security+Setup

Mukherjee, J. (ei pvm). *Atlassian.com*. Haettu 9. 1. 2020 osoitteesta https://www.atlassian.com/continuous-delivery/pipeline

Pittet, S. (a). *Atlassian.com*. Haettu 20. 12. 2019 osoitteesta https://www.atlassian.com/continuous-delivery/continuous-integration/how-to-get-to-continuous-integration

Pittet, S. (d). *Atlassian.com*. Haettu 8. 1. 2020 osoitteesta https://www.atlassian.com/continuous-delivery/continuous-deployment

Rehkopf, M. (b). *atlassian.com*. Haettu 14. 1. 2020 osoitteesta https://www.atlassian.com/continuous-delivery/principles

Smartbear. (ei pvm). *smartbear.com*. Haettu 10. 1. 2020 osoitteesta https://smartbear.com/product/testcomplete/integrations/automated-testing-with-jenkins/

Travis CI. (a). *travis-ci.com*. Haettu 17. 1. 2020 osoitteesta https://travis-ci.com/

Travis Ci. (b). *travis-ci.com*. Haettu 17. 1. 2020 osoitteesta https://travis-ci.com/plans

Travis CI. (c). *docs.travis-ci.com*. Haettu 17. 1. 2020 osoitteesta https://docs.travis-ci.com/user/docker/

Travis CI. (d). *docs.travis-ci.com*. Haettu 17. 1. 2020 osoitteesta https://docs.travis-ci.com/user/travis-ci-for-private/

Käytetyt kuvat

Kuva 1. continuos_integration_illustration-1. Haettu 14. 1. 2020 osoitteesta https://deploybot.com/assets/blog/_normal/continuos_integration_illustration-1.png

Kuva 2. TestingTriangle. Haettu 8. 1. 2020 osoitteesta http://4.bp.blogspot.com/-lgR0Ph-CLnE/UZ9CJXaaRlI/AAAAAAAAAyY/D0zkoc3KnVA/s1600/TestingTriangle.png

Kuva 3. ci cd asset updates.007. Haettu 14. 1. 2020 osoitteesta https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Kuva 4. containersvsvirtualmachines. Haettu 14. 1. 2020 osoitteesta https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/containerized-microservices-images/containersvsvirtualmachines.png