Rostedt Jarno

OHJELMISTOTYÖN TEHOSTAMINEN AUTOMAATIOYMPÄRISTÖSSÄ. CASE: JENKINS

Tietojenkäsittelyn koulutusohjelma
2018

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

OHJELMISTOTYÖN TEHOSTAMINEN AUTOMAATIOYMPÄRISTÖSSÄ. CASE: JENKINS

Rostedt, Jarno
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Huhtikuu 2018
Ohjaaja: Nieminen, Hans
Sivumäärä: 29

Asiasanat: Jenkins, jatkuva integraatio, jatkuva toimitus, ansible

---

Tämän opinnäytetyön tarkoituksena oli toteuttaa Codecontrol Oy:lle avoimeen lähdekoodiin perustuvan web-pohjaisen Jenkins-palvelinsovelluksen asentaminen toimintakuntoon fyysiselle palvelimelle ja jakaa se usealle virtuaalipalvelimelle nykyisen yhden sijasta saaden kokonaisuus hyödyntämään master-slave -arkkitehtuurin mukaista ohjelmistoautomatisointimallia.

Ennen varsinaista toteutusosuutta tämän opinnäytetyön teoriaosuudessa kerrotaan jatkuvasta integraatiosta ja jatkuvasta kehityksestä sekä niiden merkityksestä ohjelmistokehitykselle. Teoriaosuuden lopuksi esitellään toteutuksessa asennettava web-pohjainen palvelinsovellus Jenkins.

JENKINS AS PART OF THE CONTINUOUS INTEGRATION OF SOFTWARE DEVELOPMENT

Rostedt, Jarno
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Business Information Technology
April 2018
Supervisor: Nieminen, Hans
Number of pages: 29

Keywords: Jenkins, continuous integration, continuous delivery, ansible

---

The purpose of this thesis was to implement an open source web-based Jenkins server application installation into physical server for Codecontrol Oy and distribute it to multiple virtual servers instead of the existing one, thus enabling the whole utilization of the master-slave -architecture-based software automation model.

Before the actual implementation part, the theoretical part of this thesis tells about continuous integration and continuous development and their significance for software development. Web-based server application Jenkins is introduced at the end of the theoretical part.

SISÄLLYS

TERMIT JA LYHENTEET

Apache Ant	Java-pohjainen koontityökalu
CD	Continuous delivery. Jatkuva toimitus
CI	Continuous integration. Jatkuva integraatio
Instanssi	Esiintymä tai ilmentymä. Olio-ohjelmoinnissa luokan edustaja
Jenkins	Avoimeen lähdekoodin perustuva web-pohjainen palvelinsovellus
Playbook	Ansiblen konfiguraatiotiedosto
SSH	Salattuun tietoliikenteeseen tarkoitettu protokolla
WAR	Web Application Archive. Pakattu tiedosto, joka sisältää toimivan java-pohjaisen web-applikaation
XML	Extensible Markup Language. Merkintäkieli
XP	Extreme Programming. Ketterä ohjelmistokehitysmenetelmä

1 JOHDANTO

Suoraviivaisuus ja nopea palaute ovat nykypäivän ohjelmistokehityksen avainsanoja. Koodia voi syntyä alan yrityksessä päivittäin jopa tuhansia rivejä, jotka integroidaan versionhallinnan avulla valmistuvan ohjelmiston jo olemassa olevaan toimivaan koodikantaan. Miten siis voidaan varmistaa ajoissa nopeasti ja automatisoidusti jokaisen tehdyn muutoksen jälkeen, että ohjelmisto on varmasti toimiva, täyttää laadulliset vaatimukset ja on aina valmiina julkaistavaksi haluttuun toimintaympäristöön?

Ketterien kehitysmenetelmien saavuttaessa ohjelmistotuotannossa vakiintuneen aseman 2000-luvun alussa syrjäyttäen samalla perinteiset ohjelmistokehitysmenetelmät, myös jatkuvasta integraatiosta ja jatkuvasta toimituksesta tuli olennainen osa ohjelmistotuotantoa. Automatisoitua ketterää ohjelmakehitystoimintamallia sovelletaankin lähes jokaisessa nykypäivän ohjelmistoyrityksessä.

Codecontrol Oy on vuonna 2014 perustettu porilainen ohjelmisto- ja konsultointiyritys, joka tarjoaa palveluinaan muuan muassa eri verkkoalustoille verkkosivujen ja applikaatioiden toteutuksia, tietokantaratkaisuja, desktop-sovelluksia, koulutusta, auditointia sekä konsultointia. Web-pohjainen palvelinsovellus Jenkins on tärkeä osa yrityksen ketteriä ohjelmistokehitysmenetelmiä sekä käytössä yrityksen päivittäisessä ohjelmistokehityksessä.

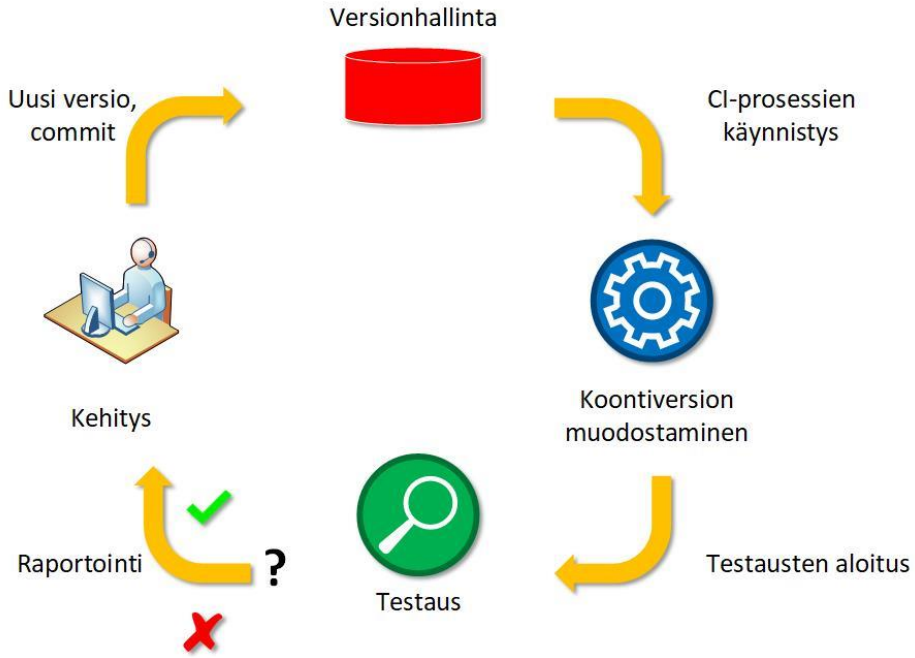
Toimeksiantonani tässä opinnäytetyössä oli asentaa Ansible-ohjelmiston avulla Codecontrol Oy:n Jenkins toimimaan nykyisen pilvipalvelupohjaisen Microsoft Azuren yhden virtuaalikoneen mallista ajettavaksi yhdelle fyysiselle palvelinkoneelle ja jakaa sen toiminta kolmelle eri virtuaalikoneelle. Nämä virtuaalikoneet määriteltiin toimimaan master-slave -arkkitehtuurin mukaisesti hajautetun koontikokonaisuuden muodostamiseen. Tästä saatu hyöty nopeuttaa yrityksen toimitusputken mukaisia koonti ja testausautomaatioprosesseja.

Ennen varsinaista toimeksiannon toteutusta perehdytään jatkuvaan integraatioon ja jatkuvaan toimitukseen kertoen niiden toiminnasta ja eri osa-alueista. Neljännessä luvussa lopuksi esitellään Jenkins ja kerrotaan sen tärkeimmistä ominaisuuksista.

2 JATKUVA INTEGRAATIO

Jatkuva integraatio (eng. continuous integration) on ketterää ohjelmakehitystoimintamallia tukeva käytäntö, jossa jokaisen kehitystiimin jäsenen kirjoittaman koodimuutoksen tai uuden ominaisuuden implementoinnin vienti käytössä olevaan versionhallintaan tehdään mahdollisimman usein, vähintään päivittäin, mikä johtaa useisiin integrointikertoihin päivässä. (Fowler 2006.)

Jokainen muutos versionhallintaan käynnistää automatisoidusti jatkuvan integraation ohjelmistotyökalun toimesta prosesseja, joilla ohjelmistosta rakennetaan uusi toimiva versio, ajetaan tarvittavat testit sitä vasten ja ilmoitetaan välittömästi kehittäjille, onnistuivatko prosessit (Kuva 1). Tämä mahdollistaa sen, että mahdolliset virheet ovat havaittavissa mahdollisimman nopeasti ja että saatavilla on aina toimiva ja laatukriteerit täyttävä versio ohjelmistosta. (Haikala & Mikkonen 2011, 175.)



Kuva 1. Jatkuvan integraation perusajatus

2.1 Historia

Ennen jatkuvaa integraatiota perinteiseen ohjelmistokehitykseen kuului, että jokainen kehitystiimin jäsen otti käyttöönsä versionhallinnasta jonkin version ohjelmasta oman

työnsä pohjaksi. Koodimuutoksen tai lisäyksen valmistuttua, se päivitettiin määrätyssä integraatiovaiheessa uusimpaan versionhallinnasta löytyvään versioon ja testattiin senhetkistä toteutusta vasten. Ongelmaksi kuitenkin koitui yhteensovittaminen muiden kehittäjien tekemien muutosten kanssa samoihin osiin ohjelmaa. Nämä aiheuttivat usein integraatiokonflikteja, toimitusviivästyksiä, ylimääräisiä kustannuksia ja pahimmassa tapauksessa työn aloittamisen alusta. (Haikala & Mikkonen 2011, 175.)

Ketterien ohjelmistokehitysmenetelmien yleistyessä 1990-luvun lopulla jatkuvasta integraatiosta tuli yksi Extreme Programming (XP) –ohjelmistokehitysmenetelmän alkuperäisistä 12:sta käytänteistä, joiden toimintatapa perustuu useisiin toistuviin ohjelmistojulkaisuihin ja lyhyisiin kehityssykleihin. (Tutorialspoint. 2018.)

2.2 Osa-alueet

Jatkuva integraatio koostuu pääasiallisesti versionhallinnasta ja automatisoidusta koostamisesta sekä testauksesta. Seuraavissa luvuissa kerrotaan näistä vielä hieman tarkemmin.

2.2.1 Versionhallinta

Versionhallinnan käyttötarkoituksena on varmistaa, että ohjelmiston kehitystä pystytään hallitsemaan vakaasti ja seuraamaan jatkuvasti sen versiointia kehityskaaren aikana. Tämä mahdollistaa esimerkiksi sen, että tiedostot voidaan palauttaa aikaisempaan versioon vääränlaisten muutosten sattuessa. (Fowler. 2006.)

Versionhallinnan merkitys korostuu projekteissa, joissa työskentelee useita kehittäjiä samanaikaisesti monien eri tiedostojen kanssa, koska ohjelmistoprojektin kaikki tiedostot säilyvät yhdessä paikassa ja ovat kaikkien kehittäjien saatavilla. (Fowler. 2006.)

Lukuisat versionhallintajärjestelmät tukevat myös monipuolisesti useiden uusien haarojen (eng. branch) luomista ohjelman päähaarasta. Monihaarautuminen mahdollistaa esimerkiksi virhekorjausten testaamisen, kokeellisten muutosten tekemisen tai uusien

ominaisuuksien lisäämisen koskematta itse ohjelman päähaaraan. On kuitenkin suositeltavaa pitää uusien haarojen lukumäärä pienenä ongelmien välttämiseksi. (Fowler. 2006.)

2.2.2 Koontiversion muodostaminen

Jatkuvassa integraatiossa koontiversion muodostaminen (eng. build) tarkoittaa lähdekoodin hakemista versionhallinnasta ja sen kääntämistä ajettavaan muotoon. Koonti suoritetaan asennettujen koontityökalujen avulla, joita ohjataan niille luoduilla koontiskripteillä (eng. build scripts). Esimerkiksi Java-pohjaista Apache Ant –koontityökalun toimintaa ohjataan XML-muotoisilla koontisktripteillä. (Duvall. 2007. 256.)

Koontiskriptien määrittelyn ei tarvitse kuitenkaan rajoittua pelkästään lähdekoodin kääntämiseen, vaan ne voivat sisältää tämän lisäksi myös koko jatkuvan integraation läpikäynnin, kuten esimerkiksi komennot testausten ja tarkastelujen ajamiseen, integroinnin tietokantaan sekä ohjelman saattamisen toimivaksi ohjelmaksi toimintaympäristöön. On kuitenkin huomioitava, että liian monen eri prosessin lisääminen samaan koontiskriptiin saattaa hidastaa lopullisen palautteen muodostumista. (Duvall. 2007. 67-68.)

Koontiversioiden muodostamisen tulisi myöskin olla alusta loppuun automatisoitua, joka on mahdollista käynnistää tarvittaessa myös yhden napin painalluksella. (Duvall. 2007. 68.)

2.2.3 Testaus

Ohjelmassa esiintyvien mahdollisten bugien, yhteensovittamis- tai syntaksivirheiden välttämiseksi automaattinen testaus on nykypäivänä keskeinen osa jatkuvaa integraatiota. Ohjelma voidaan ajaa suoraan koonnin jälkeen, mutta ilman automatisoitujen testien ajamista ohjelman täydelliseen toimivuuteen ei voida luottaa. (Fowler. 2006.)

Automatisoituun testaukseen jatkuvan integroinnin järjestelmässä voidaan sisällyttää esimerkiksi yksikkötestien, komponenttitestien, järjestelmätestien, lataus- ja suorituskykytestien sekä turvallisuustestien ajaminen, joka nopeuttaa koontiversion muodostamista. (Duvall. 2007. 15.)

Monet ohjelmistokehittäjät käyttävät jatkuvan integroinnin järjestelmien yhteydessä testaustyökaluina esimerkiksi JUnit-, NUnit- tai muita XUnit -ohjelmistokehyksiä (eng. framework). (Duvall. 2007. 15.)

Jatkuvaan integraatioon kuuluvaa testausautomatisointia ei kuitenkaan tule ottaa täysin itsestäänselvyytenä, sillä automatisointi ei poista täysin käsin testaamisen tarpeellisuutta. On totta, että testausautomatisointi vapauttaa resursseja, karsii kustannuksia sekä vähentää käsin tehtävien regressiotestausten määrää, mutta sen tehokas hyödyntäminen vaatii ylläpitoa ja resursseja. (Kasurinen. 2013. 76-77.)

3 JATKUVA TOIMITUS

Jatkuvan integraation käyttöönotto on iso askel kohti tuottavuutta ja laadun varmistamista useimmille ohjelmistoprojekteille. Se varmistaa, että ohjelmistotiimi voi tehdä laajoja ja monimutkaisia järjestelmiä, jotka ovat luotettavia ja helposti hallittavia. Kuitenkaan yksinään jatkuva integraatio ei pelkästään riitä täydellisen jatkuvan ja ketterän ohjelmistokehityksen saavuttamiseksi. (Humble, J. & Farley D. 105.)

Jatkuva toimitus (eng. continuous delivery) on jatkuvan integraation tapaan ketterää ohjelmakehitystoimintamallia tukeva käytäntö sekä toiminnallinen jatke jatkuvalle integraatiolle, joka mahdollistaa toimivan ja halutun ohjelmistoversion julkaisemisen automatisoidusti kehitysympäristöön, tuotantoympäristöön asiakkaalle tai tuotantoympäristöä vastaavaan ympäristöön missä vaiheessa tahansa kesken kehitystä. (Fowler. 2013.)

3.1 Toimitusputki

Jatkuvan toimituksen käyttöönotto edellyttää toimitusputken (eng. delivery pipeline) käyttöönottoa osaksi ohjelmistokehitystä. Toimitusputki jakaa koontiversion muodostamisen eri osioihin (Kuva 2), joissa varmistetaan ja validoidaan muutokset sekä ennaltaehkäistään virheiden muodostuminen lopullisessa ohjelmistoversiossa. Ohjelmistosuoritusympäristöön julkaisu on yleensä viimeinen taso toimitusputkessa. (Fowler. 2013.)

Kuva 2. Esimerkki toimitusputkesta ja siihen kuuluvista osioista (Humble, J. & Farley D. 2011.)

Toimitusputki toimii virhetilanteiden sattuessa myös jatkuvan integraation tavoin. Toimitusputken tietyn osion havaitessa virheen koontiversion testauksessa tai muodostuksessa koontiversion muodostaminen perutaan toimitusputkessa ja ilmoitetaan virheestä välittömästi kehittäjille. (Mukherjee. 2018.)

3.2 Jatkuvan toimituksen ja jatkuvan julkaisun ero

Jatkuva toimitus saatetaan sekoittaa jatkuvaan julkaisuun (eng. continuous deployment), joka tarkoittaa valmiiden koontiversioiden julkaisemista automaattisesti tuotantoympäristöön. Tämä johtaa siihen, että tuotantoympäristöön suunnattuja ohjelmistoversioita syntyy päivässä useita, kun taas jatkuvassa toimituksessa ohjelmistoversioita syntyy silloin, kun siihen on tarvetta. (Fowler. 2013.)

4 JENKINS

Jenkins on markkinoiden johtava avoimeen lähdekoodiin perustuva Java-pohjainen alun perin jatkuvaan integraation tarkoitettu, mutta myös myöhemmin jatkuvaan toimitukseen luotu web-pohjainen palvelinsovellus, jota käytetään monessa erilaisessa ohjelmistokehitystiimin koosta riippumatta olevissa ohjelmistoprojekteissa. Jenkinsin ohjelmistokielituki on myös hyvin laaja ja se tukee esimerkiksi Python-, Java-, Ruby-, Groovy-, Grails- ja PHP-kieliä sekä lukuisia eri skriptikieliä. (Smart. 2011. 3.)

Suuren suosion saavuttaneen Jenkinsin takana on sen helppo käyttöönotto, nopea oppimiskäyrä, laajennettavuus lukuisten erilaisten ilmaisten lisäosien avulla sekä suuren ja aktiivisen yhteisön tuki mahdollisten kysymysten ja ongelmien kohdatessa. (Smart. 2011. 3.)

4.1 Historia

Jenkinsin historia alkaa vuodesta 2004, jolloin ohjelmistokehittäjä Kohsuke Kawaguchin työskennellessään Sun Microsystemsillä aloitti harrastusprojektinaan kehittämään Jenkinsiä nimellä Hudson. Hudsonin kehittyessä vuosien varrella Sun Microsystems alkoi hyödyntää ensin sitä myös yrityksen omissa projekteissa ja myöhemmässä vaiheessa omana tuotteena. Vuoteen 2010 mennessä Hudsonista oli tullut jatkuvan integroinnin johtava ratkaisu 70 % markkinaosuudella. (Smart. 2011. 4.)

Vuonna 2009 Oraclen ostettua Sun Microsystemsin alkoivat ongelmat Hudsonin kehitysyhteisön ja Oraclen välillä johtuen Java.netin infrastruktuurista, Oraclen omistusoikeudesta Hudsoniin sekä kehitysprosesseihin liittyvistä työtavoista. Hudsonin kehitysyhteisö Kohsuken johdolla halusi pitää kehityksen avoimena ja yhteisökeskeisenä, joka oli aikaisemmin todettu toimivaksi ratkaisuksi, kun taas Oracle ajoi tarkkaan ohjattua kehitysmallia hitaammalla julkaisuaikataululla. (Smart. 2011. 4.)

Näiden ristiriitojen seurauksena Hudsonin kehitysyhteisö siirsi vuonna 2011 alkuperäisen Hudsonin koodikannan uuteen GitHub-versionhallinnan tietovarastoon ja muutti Hudsonin nimen Jenkinsiksi. Tämän jälkeen Oracle jatkoi Hudsonin kehitystä

Jenkinsin eriytymisen jälkeen Nexus -ohjelmistoista tutun SonaTypen kanssa, mutta lopetti sen ylläpidon vuoden 2017 aikana. (Smart. 2011. 4.)

4.2 Järjestelmävaatimukset

Jenkinsin käyttöympäristön järjestelmävaatimuksien arviointi riippuu monesta eri tekijästä, jotka on mietittävä tarkasti lopullisen käyttöympäristön optimaalisen konfigroinnin saavuttamiseksi. Yhdeksi suurimmaksi haasteeksi Jenkinsin käytön aloituksessa on juurikin muodostunut yrityksen nykyisten ja tulevaisuuden tarpeiden arviointi. (Jenkins a.)

Jenkins tukee master/agent-arkkitehtuurin mukaista toimintatapaa, jossa koontiversioiden luomisesta kertyvää työkuormaa jaetaan yhden Jenkins-instanssin omaavalta master-järjestelmältä usealle eri slave-järjestelmälle (Luku 4.7). Keskusmuistin allokoinnin määrä yhdelle omatoimiselle master-järjestelmälle omassa käyttöympäristössä voi vaihdella pienimillään 200 megatavusta aina yli 70 gigatavuun riippuen myös siitä, kuinka paljon yrityksen ohjelmistoprojektien koontiprosesseihin on laskettu tarvitsevan tehoa. (Jenkins a.)

Slave-järjestelmien laitevaatimusten määrittely saattaa olla myös haastavaa ja aikaa vievää riippuen slave-järjestelmien lukumäärästä automaatiossa. Yleisenä sääntönä voidaan pitää kuitenkin sitä, että asennettavat slave-järjestelmät ovat laitetehoiltaan samanlaisia ja räätälöity tekemään yhtä tiettyä työtä. (Jenkins a.)

4.3 Asennusvaihtoehdot

Riippuen yrityksen tarpeista Jenkinsiä asennettassa on valittavissa kaksi versiota - Weekly-versio ja LTS (Long-Term Support) -versio. Weekly-versio päivittyy viikoittain uusimmilla virheenkorjauspäivityksillä ja uusilla ominaisuuksilla, joka mahdollistaa sen, että Jenkins pysyy aina päivitettynä uusimpaan versioonsa. (Jenkins b.)

LTS-versiossa päivitykset toimitetaan harvemmin, yleensä kolmen kuukauden välein ja vain mahdolliset kriittiset virheenkorjaukset toimitetaan tarvittaessa ennen varsinaista uutta julkaisua. Tätä versiota on suositeltavampaa käyttää sen vakaamman toiminnan ja pienemmän päivitystahdin tuomien etujen vuoksi. (Smart. 2011. 4)

4.4 Asennusympäristöt

Jenkinsiä voidaan ajaa WAR-tyyppisenä pakettitiedostona millä tahansa käyttöjärjestelmällä tai alustalla, johon Java on asennettavissa. Vaihtoehtoisesti Windows- ja Mac OS X –käyttöjärjestelmiin asennus on mahdollista tehdä niille suunnattujen asennuspakettien avulla. Linux-jakeluversioihin ja Unixiin pohjautuviin käyttöjärjestelmiin asennus voidaan tehdä niiden omien komentorivipohjaisten paketinhallintaohjelmistojen avulla. (Jenkins b.)

Paikallisten käyttöjärjestelmäasennusten lisäksi Jenkins voidaan ottaa käyttöön Microsoft Azure -pilvipalvelussa tai Docker-levykuvana Docker-konttialustalla. Docker-asennuksessa Javan ajo- tai kehitysympäristön asennusta ei vaadita (Jenkins b.) Taulukosta 1 voidaan vielä tarkastella Jenkinsiä virallisesti tukevia käyttöjärjestelmiä, asennustapoja sekä tarjottuja versioita.

Taulukko 1. Jenkinsiä virallisesti tukevat käyttöjärjestelmät, asennustavat ja tarjotut versiot

Käyttöjärjestelmä	Asennus-ohjelma	Paketinhallin-taohjelmisto	WAR	LTS	Weekly
Arch Linux		X	X		X
Debian-pohjaiset Linuxit		X	X	X	X
Fedora-pohjaiset Linuxit		X	X	X	X
FreeBSD		X	X	X	X
Gentoo Linux		X	X	X	X
Mac OS X	X	X	X	X	X
OpenBSD		X	X	X	X
OpenIndiana Hipster		X	X		X
openSUSE		X	X	X	X
Windows	X		X	X	X

4.5 Laajennettavuus

Yksi Jenkinsin ydinosa on laajennukset (eng. plugins), jotka mahdollistavat Jenkinsin mukauttamisen tai ominaisuuksien laajentamisen lukuisiin erilaisiin käyttötarkoituksiin – oli sitten kyse esimerkiksi versionhallinnasta, koontityökaluista, koontiversioiden valmistumisen ilmoittamisesta, koodin laadullisesta mittaamisesta tai integroinnista ulkoisten järjestelmien kanssa. (Smart. 2011. 3)

Liittännäisten asennus Jenkinsiin on myös helppo tehdä Jenkinsin oman graafisen käyttöliittymän avulla suoraan liitännäishallinnasta tai sisäänrakennetun komentorivikäyttöliittymän (Jenkins CLI) kautta antamalla liitännäisten asennukseen vaadittuja komentoja. (Jenkins c.)

4.6 Jenkins Pipeline

Jenkins Pipeline on kokoelma laajennuksia, jotka mahdollistavat jatkuvan toimituksen mukaisen toimitusputken käyttöönottamisen Jenkinsissä. Jenkins Pipelinen määritys tehdään Jenkinsfile –nimiseen tekstitiedostoon, jossa määritellään jatkuvan toimituksen kokonaisvaltainen automaatioprosessi ohjelman koonnista testaukseen ja toimitukseen ja sitä voidaan lukea joko paikallisesti tai versionhallinnasta. (Jenkins d.)

Tiedoston kirjoittamiseen voidaan käyttäen kahta erilaista syntaksitapaa – deklaratiivista (eng. declarative) tai skriptattua (eng. scripted). Deklaratiivisella tavalla määritelty Jenkinsfile-tiedosto on kirjoitusasultaan ja luettavuudeltaan helppolukuisempi tarjoten paremmat syntaktiset ominaisuudet verrattuna skriptatulla tavalla määriteltyyn Jenkinsfile-tiedostoon. Skriptattu toimitusputki kirjoitetaan dynaamista Apache Groovy -ohjelmointikieltä käyttäen. (Jenkins e.)

Jenkinsfile-tiedoston mukainen toimitusputki voidaan luoda kolmella eri tavalla:

1. Käyttäjäystävällisesti graafisella Blue Ocean –lisäosalla, jolla Jenkinsfile-tiedosto tallennetaan versionhallinnan tietovarastoon päivittyen automaattisesti joka kerta, kun Blue Oceanissa tehdään muutoksia toimitusputkeen

2. Suoraan Jenkinsin klassisesta käyttöliittymästä sisäänrakennetulla skriptieditorilla, jolloin Jenkinsfile tallentuu samaan kansioon, johon Jenkins on itse asennettu
3. Itse manuaalisesti, jos Blue Oceanin tai klassisen käyttöliittymän mukaiset luontitavat eivät riitä. Tällöin Jenkinsfile-tiedosto luodaan alusta loppuun manuaalisesti tekstieditorissa ja tallennetaan se versionhallinnan tietovarastoon

4.7 Skaalautuvuus master-slave -arkkitehtuurilla

Jenkins tukee master-slave -arkkitehtuuria hajautetun koontikokonaisuuden muodostamiseen. Tällä tarkoitetaan sitä, että automatisoidussa suoritusympäristössä yksi Jenkinsin omaava palvelinkone toimii master-järjestelmänä, joka käsittelee esimerkiksi koontiversioiden muodostamisen aikataulutuksen, kääntö- ja testaustöiden jakamisen useille slave-järjestelmille niiden varsinaiseen suorittamiseen, slave-järjestelmien valvomisen sekä koontiversioiden muodostamisesta syntyvien tulosten tallentamisen ja raportoinnin. Tämä toimintapa vapauttaa master-järjestelmän resurssikuormaa koontiympäristön parempaan hallitsemiseen ja nopeuttaa koontiversioiden muodostamista. (Smart. 2011. 301-302.)

Termi slave viittaa pieneen suoritettavaan Java-tiedostoon, jota ajetaan automaattisesti etäjärjestelmässä käyttöjärjestelmästä riippumatta ja joka kuuntelee koontiversion muodostamiseen liittyviä prosessipyyntöjä master-järjestelmältä TCP/IP-protokollan kautta. Slave-järjestelmä voidaan määrittää suorittavan ainoastaan tiettyjä tehtäviä tai antaa Jenkinsin valita master-järjestelmässä seuraavan vapaan slave-järjestelmän suorittamaan tiettyä tehtävää. (Smart. 2011. 302.)

5 TOTEUTUKSESSA KÄYTETYT OHJELMAT

Toteutuksessa käytettiin pääasiallisesti kolmea eri ohjelmaa. Paikallisten virtuaalikoneiden luontiin ja hallintaan käytettiin Oracle VM VirtualBox –virtualisointiohjelmaa ja asennuksen kokonaisvaltaiseen automatisointiin Ansiblea, jolle luotiin Visual Stu-

dio Code-tekstieditorilla ennen asennuksen käynnistämistä tarvittavat määritystiedostot, jotta asennus saatiin tapahtumaan oikeaoppisesti. Seuraavissa kappaleissa kerrotaan näistä ohjelmista vielä lyhyesti.

5.1 Ansible

Ansible on IT-automaatio-ohjelma, jonka avulla voidaan konfiguroida palvelinjärjestelmiä, asentaa ohjelmistoja ja orkestroida vaativampia IT-tehtäviä, kuten jatkuvia toimituksia tai päivityksiä, jotka eivät vaadi erikseen ohjelmiston alasajoa. Ansiblen suorittamia tehtäviä ja rooleja hallitaan ja ajetaan YAML-tyyppisillä Playbook-tiedostoilla. (Ansible a. 2017.)

Rooleilla (eng. roles) Ansiblessa voidaan ryhmittää samaan aihealueeseen kuuluvat tehtävät samaan paikkaan. Roolien avulla voidaan siis useiden eri tehtävien suoritus automatisoida helposti. Tämän lisäksi roolit ovat myös uudelleenkäytettäviä, joten myös muiden käyttäjien tekemiä valmisrooleja on mahdollisuus ladata ilmaiseksi Ansiblen omasta Ansible Galaxy -yhteisöpalvelusta ja ottaa käyttöön omissa asennusprojekteissa (Ansible b. 2018.).

5.2 Oracle VM VirtualBox

Oracle VM VirtualBox on Windows-, Mac OS X-, Linux- ja Oracle Solaris –käyttöjärjestelmille suunnattu ilmainen ja avoimeen lähdekoodiin perustuva virtualisointiohjelma, jonka avulla voidaan ajaa virtuaalisesti useita eri vieraskäyttöjärjestelmiä samaan aikaan (VirtualBox.).

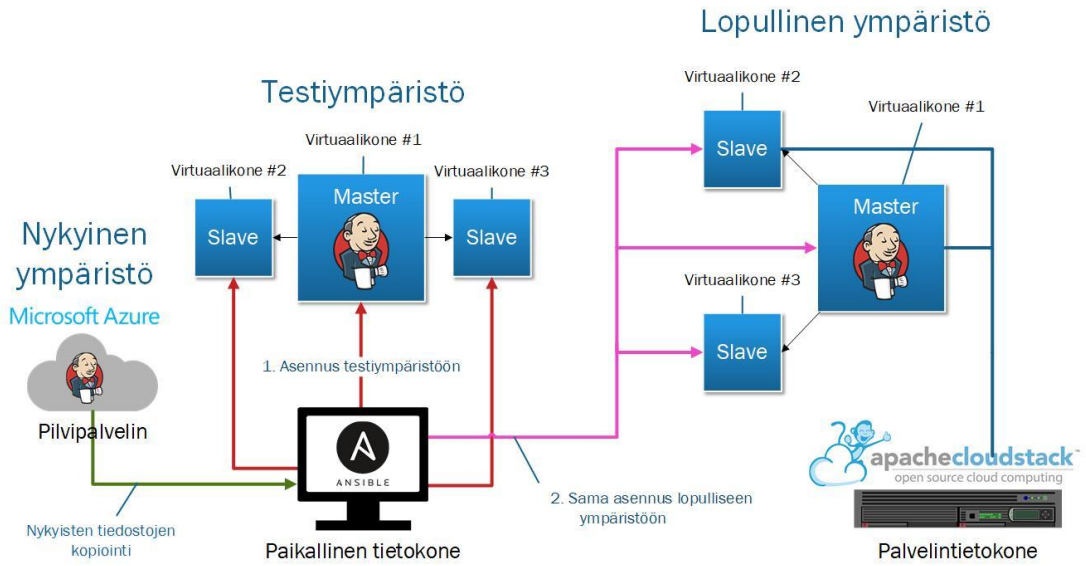
5.3 Visual Studio Code

Visual Studio Code on Windows-, Mac OS X- ja Linux -käyttöjärjestelmille suunnattu kevyt ja ilmaiseksi ladattava avoimeen lähdekoodiin perustuva tekstieditori, joka tukee lähes jokaista nykypäivän ohjelmointikieltä ja jonka ominaisuuksia voidaan laajentaa lukuisilla eri lisäosilla (Visual Studio Code. 2018.).

6 TOIMEKSIANNON TOTEUTUS

Asennus suunniteltiin toteutettavaksi niin, että lopullista toimintaympäristöä vastaava ympäristö luodaan aluksi paikallisella tietokoneella kolmelle eri virtuaalikoneelle ja testataan sen toimivuus ennen asennuksen tekemistä fyysisen palvelimen Apache CloudStack –järjestelmään pohjautuviin virtuaalikoneisiin.

Nykyisen Azuressa toimivan Jenkins-instanssin käytössä olevat määrittely- ja asetustiedostot kopioidaan myös paikalliselle koneelle ja puretaan master-virtuaalikoneen Jenkins-instanssiin, jotta voidaan välttyä nykyisen Jenkins-instanssin käytössä olevien työ- ja testausvaiheiden uudelleenmäärittämiseltä. Suunniteltua prosessikokonaisuutta voidaan tarkastella alla olevassa kuvassa (Kuva 3).



Kuva 3. Suunniteltu prosessikokonaisuus

6.1 Virtuaalikoneiden asennus

Työ aloitettiin määrittelemällä valmiiksi kolme uutta virtuaalikonetta Oracle VM VirtualBox -virtualisointiohjelmistossa. Jokaiselle virtuaalikoneelle annettiin kuvaava nimi ja asennettavan käyttöjärjestelmän tyyppi. Tässä tapauksessa virtuaalikoneiden nimeksi annettiin helposti tunnistettavat nimet JenkinsMaster, JenkinsSlave1 ja JenkinsSlave2. Käyttöjärjestelmäksi oli syytä valita 64-bittinen Ubuntu Server Linux,

koska nykyistä Jenkins-instanssia ajettiin Azuressa myös Ubuntu Server -virtuaalikoneessa.

Virtuaalikoneiden laiteresurssien määrittelyssä keskusmuistin kokoa ei tällä kertaa tarvinnut säätää erikseen, vaan suositeltu yhden gigatavun muisti todettiin olevan riittävä. Kovalevytilaa jokaiselle virtuaalikoneelle annettiin 20 gigatavua, jonka laskettiin olevan sopiva määrä Jenkinsin toimintaympäristölle.

Oracle VM VirtualBox-virtualisointiohjelmiston alkumääritysten jälkeen Ubuntu Server -käyttöjärjestelmän oma asennus aloitettiin virtuaalikoneisiin. Asennuksien aikana ei ollut tarpeen tehdä erityisiä määrityksiä käyttöjärjestelmälle, kuten asentaa graafista käyttöliittymää (GUI) tai osioida käytettävää levytilaa, joten asennus tehtiin suurilta osin automatisoidusti.

6.2 Yhteyden muodostaminen fyysisestä koneesta virtuaalikoneisiin

Virtuaalikoneiden asennusten jälkeen oli tarpeen luoda yhteys paikallisen tietokoneen käyttöjärjestelmästä virtuaalikoneisiin SSH-protokollan avulla, jotta kirjautuminen ja hallinnointi sekä Ansiblen suorittaminen paikallisen- ja virtuaaliympäristön välillä olisi mahdollista.

SSH-protokollamääritystä varten virtuaalikoneiden IP-osoitteet muutettiin dynaamisesta kiinteäksi ja kirjattiin ylös. Tämän jälkeen oli vielä tarpeen tarkistaa, että portti 22 oli auki ja tarvittaessa avata se jokaisessa virtuaalikoneessa, koska OpenSSH-ohjelmisto kuuntelee portti 22:n välityksellä tulevaa ja lähtevää verkkoliikennettä.

Paikallisen tietokoneen puolella Oracle VM VirtualBox -virtualisointiohjelmistossa tehtiin jokaiselle virtuaalikoneelle uusi TCP-protokollan mukainen porttiohjaussääntö, joihin määriteltiin tarpeelliset arvot SSH-yhteyden käyttöönottamiseen.
Alla olevasta taulukosta voidaan tarkastella vielä tarkemmin virtuaalikoneiden porttiohjaussääntöjen arvoja (Taulukko 1).

Taulukko 1. Porttiohjaussäännöt virtuaalikoneille Oracle VM VirtualBoxissa

Virtuaa-likone	Proto-kolla	Paikallisen tietokoneen localhost-osoite	Virtuaa-likoneen portti	Virtuaalikoneen IP-osoite	SSH-protokollan portti
Jenkins-Master	TCP	127.0.0.1	2222	10.0.2.15	22
Jen-kinsS-lave1	TCP	127.0.0.1	2223	10.0.2.16	22
Jen-kinsS-lave2	TCP	127.0.0.1	2224	10.0.2.17	22

Viimeisenä vaiheena SSH-yhteyden muodostuksessa SSH-avainpari muodostettiin paikallisella koneella ja kopioitiin jokaiseen virtuaalikoneeseen virtuaalikoneiden yksilöidyn porttinumeron ja virtuaalikoneen nimen avulla (Kuva 4).

```
$ ssh-keygen -t -rsa
$ ssh-copy-i -p 2222 jarno@jenkinsmaster
```

Kuva 4. Komennot SSH-avaimen luontiin ja kopiointiin master-virtuaalikoneeseen

6.3 Nykyisen Jenkinsin tiedostojen siirto

Azurella toimivan nykyisen Jenkins-instanssin asetus-/ja määrittelytiedostojen siirto aloitettiin seuraavaksi. Siirtäminen suunniteltiin tehtäväksi niin, että tiedostot kopioidaan ensin paikalliselle tietokoneelle ja lopuksi Ansiblelle luodun roolin avulla siirretään master-järjestelmään. Toimenpiteen aloittamiseksi paikalliselle tietokoneelta muodostettiin SSH-yhteys Azuressa toimivaan Ubuntu-virtuaalikoneeseen.

Ensimmäiseksi ajettiin Jenkinsin taustapalvelu alas ennen varsinaista tiedostojen siirtoa. Kopioitavia kohteita oli kaksi – ensimmäisenä Jenkinsin asennuskansio ja toisena versionhallintaan liitetty kansio, joka sisälsi määritystiedostot halutun ohjelmistoversion julkaisemisen käyttöympäristöön jatkuvan toimituksen mukaisesti. Siirron helpottamiseksi molemmat tiedostokansiot pakattiin kokonaisuudessaan kahdeksi eri tie-

dostoksi, jonka jälkeen ne siirrettiin fyysiselle koneelle ftp-tiedonsiirto-ohjelman välityksellä odottamaan asennusroolin määrittämistä. Lopuksi Jenkinsin taustapalvelu käynnistettiin takaisin ylös.

6.4 Jenkinsin asennus paikallisille virtuaalikoneille

Jenkinsin asennus aloitettiin paikallisella koneella Ansiblella luomalla Visual Studio Code –tekstieditorilla aluksi inventory-tiedosto, joka määrittelee isäntäkoneet ja isäntäkoneiden ryhmät joihin Playbook-tiedostot kohdistuvat. Isäntäkoneiden jakaminen ryhmiin mahdollistaa samojen operaatioiden kohdistamisen kaikille saman ryhmän isäntäkoneille. Tässä tapauksessa asennuskohteiksi määriteltiin kolme paikalliselle koneelle luotua virtuaalikonetta ja jaettiin ne omiin asennusryhmiinsä (Kuva 5).

```
jenkins-master ansible_ssh_host=127.0.0.1 ansible_port=2222
jenkins-slave-1 ansible_ssh_host=127.0.0.1 ansible_port=2223
jenkins-slave-2 ansible_ssh_host=127.0.0.1 ansible_port=2224

[all:children]
jenkins
jenkins-slaves

[jenkins]
jenkins-master

[jenkins-slaves]
jenkins-slave-1
jenkins-slave-2
```

Kuva 5. Inventory-tiedosto Jenkinsin asennusta varten

Inventory-tiedoston jälkeen Ansiblen kansioon lisättiin ansible.cfg-asetustiedosto, jolle määriteltiin omaan ympäristömuuttujaan aiemmin luodun inventory-tiedoston sijainti. Tämän ympäristömuuttujan avulla Ansiblelle kerrotaan mille virtuaalikoneille se voi olla yhteydessä.

6.4.1 Roolit

Asennukseen ja Jenkinsin nykyisen toimintamallin jatkamiseen tarvittavia rooleja ei ollut kaikkia tarvetta lähteä rakentamaan jokaista itse vaan osa voitiin asentaa valmisrooleina. Valmisroolien määrittäminen tehtiin requirements.yml-tiedostoon, jonka avulla tarvittavat valmisroolit voitiin ladata ja asentaa samalla kerralla. Asennuksen jälkeen valmisroolit asettuivat Ansiblen kansion roles-alikansioon.

Itse luotuja rooleja oli tarve tehdä neljä - ensimmäinen nykyisen Jenkins-instanssin asetus-/ja määrittelytiedostojen siirtämiseen ja purkamiseen, toinen Robot Framework-testiautomaatiotyökalun asennukselle, kolmas .NET Framework-ympäristön asennukselle sekä viimeinen slave-järjestelmien asennukselle ja aktivoimiselle.

Slave-järjestelmiin tarkoitettua roolia varten luotiin myös kaksi roolin suorittamisen aikana ajettavaa sh-skriptitiedostoa, joista ensimmäisen skriptin avulla slave-järjestelmät lisätään automaattisesti osaksi master-järjestelmässä toimivaa Jenkinsiä ja toisen skriptin avulla slave-järjestelmien ajo käynnistetään. Asennuksiin tarvittavia rooleja voidaan tarkastella vielä tarkemmin alla olevasta taulukosta (Taulukko 2).

Taulukko 2. Tarvittavat roolit Jenkinsin kokonaisvaltaiseen asennukseen virtuaalikoneille

Rooli	Tekijä	Tehtävä	Käyttö master-järjestelmään	Käyttö slave-järjestelmiin
Ansible Jenkins CI	geerlingguy	Jenkins master-asennus	X	
Ansible Python	sunscrapers	Pythonin asennus	X	X
Ansible Docker	geerlingguy	Dockerin asennus	X	X
Xvfb	pablerass	Xvfb:n asennus	X	X
Jenkins-Master	jarno rostedt	Nykyisen Jenkinsin tiedostojen siirto	X	
Robot Framework	jarno rostedt	Robot Frameworkin asennus	X	X

.Net Core SDK/Runtime	jarno.rostedt	.Net Core 1.0.4 asennus	X	X
JenkinsSlave	jarno.rostedt	Slave-järjestelmien asennus ja aktivointi		X

6.4.2 Playbook-tiedostot

Määrittelytiedostojen ja roolien asennuksen jälkeen Ansiblen Playbook-tiedostot voitiin nyt määrittää valmiiksi asennuksia varten. Kokonaisuus suunniteltiin tehtäväksi hierarkkisesti, jolloin kolmen eri Playbook-tiedoston mukaiset asennukset voitiin käynnistää ainoastaan yhden Playbook-tiedoston avulla automatisoidusti. Playbook-tiedostoja voidaan tarkastella vielä tarkemmin alla olevasta taulukosta (Taulukko 3).

Taulukko 3. Lopulliset Playbook-tiedostot, niiden sisältö ja käyttökohteet

Playbook-tiedosto	Sisältö	Käyttökohde
jenkins.yml	Kaikkien Playbook-tiedostojen yhteinen asennus. Referenssitiedosto muihin Playbook-tiedostoihin.	Kaikki
jenkins-common.yml	Yhteisten roolien asennus kaikille koneille	Kaikki
jenkins-master.yml	- Jenkinsin asennus - Nykyisen Jenkins-instanssin tiedostojen siirto master-järjestelmään	Master-järjestelmä
jenkins-slaves.yml	Slave-koneiden asennus	Slave-järjestelmät

6.4.3 Asennuksen käynnistäminen

Jenkins.yml –tiedoston mukaiset asennukset käynnistettiin paikallisiin virtuaalikoneisiin Ansiblen omalla asennuskomennolla, jolla asennus käynnistyi automatisoidusti

Playbook-tiedostoihin määriteltyjen roolien mukaisesti. Alla olevasta kuvasta voidaan tarkastella asennuksen aloituskomentoa (Kuva 6).

```
$ ansible-playbook -i inventory jenkins.yml
```

Kuva 6. Ansiblen Playbook-tiedoston asennuskomento

Asennuksen valmistuttua Ansible ilmoitti vielä loppukoosteessa, onnistuivatko asennuksen kohdekoneisiin vai oliko tarpeen tehdä vielä korjauksia. Alla olevasta kuvasta voidaan todeta, että kaikki asennukset onnistuivat odotetusti (Kuva 7).

```
 ____________
< PLAY RECAP >
 ------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

jenkins-master             : ok=57   changed=9    unreachable=0    failed=0
jenkins-slave-1            : ok=29   changed=4    unreachable=0    failed=0
jenkins-slave-2            : ok=29   changed=4    unreachable=0    failed=0
```

Kuva 7. Ansiblen tekemä loppukooste asennuksesta

6.4.4 Jenkinsfilen muokkaus slave-järjestelmien käyttöönottoon

Onnistuneen asennuksen jälkeen versionhallinnan tietovarastossa sijaitsevaan ja käytössä olevaan Jenkinsfile-tiedostoon oli tarpeen määrittää, miten jatkuvan toimituksen mukaisen toimitusputken osioihin kuuluvat työt tulevat jakaantumaan master- ja slave-järjestelmien kesken. Ennen muutosten tekemistä Jenkinsfile-tiedoston mukaiset työt oli määritelty tehtäväksi ainoastaan yhdellä koneella.

Määrittämisen aloittamiseksi Jenkinsfile-tiedoston sisältävän versionhallinnan tietovarasto kloonattiin paikalliselle koneelle, jonka jälkeen halutut muutokset tehtiin metodipohjaisesti jokaisen osion tehtäviin Visual Studio Coden avulla. Jokaisen osion kohdalla tehtävät yritettiin jakaa tasaisesti, jotta master-slave -arkkitehtuurista saatava hyöty olisi mahdollisimman suuri.

6.5 Toimivuuden tarkastelu master-järjestelmässä

Jenkinsin käytön ja toimivuuden tarkastelun aloittamiseksi paikallisella koneella Oracle VM VirtualBox -virtualisointiohjelmistossa tehtiin uusi porttiohjaussääntö portille 8080 master-järjestelmäksi määritellylle JenkinsMaster-virtuaalikoneelle. Tämän jälkeen Jenkinsin käyttö voitiin aloittaa kirjautumalla internetselaimella osoitteeseen 127.0.0.1:8080.

Ensimmäisenä oli tarpeen tarkistaa, että slave-järjestelmät olivat aktivoituneet ja näkyivät oikein Jenkinsin käyttöliittymässä (Kuva 8). Tämän jälkeen, kun slave-järjestelmät olivat todettu toimiviksi, käytiin läpi toimitusputken eri osiot kuten hyväksyntätestaus ja kääntäminen sekä toimivan version asentaminen kehitysympäristöön testiajojen muodossa läpi. Onnistuneiden testiajojen jälkeen voitiin todeta, että kokonaisuus oli valmis asennettavaksi fyysiselle palvelimelle.

S	Name ↓	Architecture	Clock Difference	Free Disk Space
	jenkinsslave1	Linux (amd64)	In sync	N/A
	jenkinsslave2	Linux (amd64)	In sync	N/A
	master	Linux (amd64)	In sync	5.56 GB
	Data obtained	**25 min**	**25 min**	**25 min**

Kuva 8. Master- ja slave-järjestelmät Jenkinsissä

6.6 Toimivan kokonaisuuden asennus palvelimelle

Kun kokonaisuus oli todettu toimivaksi paikallisella koneella toimivissa virtuaalikoneissa, aloitettiin seuraavaksi sen asentaminen Ansiblella fyysiselle palvelinkoneelle. Palvelinkoneelle oli asennettu valmiiksi Java-pohjainen avoimen lähdekoodin IaaS-järjestelmä Apache Cloudstack, johon luotiin vielä lopulliset Ubuntu-virtuaalikoneet asennusta varten sekä muodostettiin niihin SSH-yhteydet fyysiseltä koneelta, jotta asennus onnistuisi (Kuva 9).

Name	Internal name	Display Name	IP Address	Account	Zone Name	State	Quickview
jenkins-slave-2	i-4-28-VM	jenkins-slave-2	192.168.1.101		cc-zone	Running	
jenkins-slave-1	i-4-27-VM	jenkins-slave-1	192.168.1.64		cc-zone	Running	
jenkins-master	i-4-26-VM	jenkins-master	192.168.1.70		cc-zone	Running	

Kuva 9. Lopullisen ympäristön virtuaalikoneet

Ansiblen inventory-tiedostoon muutettiin fyysisellä palvelinkoneella toimivien virtuaalikoneiden ip-osoitteet paikallisella koneella toimivien virtuaalikoneiden tilalle (Kuva 9), minkä jälkeen asennus käynnistettiin uudelleen samalla Ansiblen komennolla kuin aikaisemminkin.

```
jenkins-master ansible_ssh_host=127.0.0.1 ansible_port=2222
jenkins-slave-1 ansible_ssh_host=127.0.0.1 ansible_port=2223
jenkins-slave-2 ansible_ssh_host=127.0.0.1 ansible_port=2224
```

```
jenkins-master ansible_ssh_host=192.168.1.70
jenkins-slave-1 ansible_ssh_host=192.168.1.64
jenkins-slave-2 ansible_ssh_host=192.168.1.101
```

Kuva 10. Paikallisten virtuaalikoneiden osoitteiden muuttaminen lopullisten virtuaalikoneiden osoitteiksi Ansiblen inventory-tiedostossa

Asennuksen jälkeen viimeisenä toimenpiteenä uuden käyttöympäristön aloittamiseksi oli nykyisen Microsoft Azuressa toimivan Jenkins-instanssin poistaminen käytöstä, jotta uudella fyysisen palvelimella toimivan kokonaisuuden kanssa ei syntyisi ristiriitoja. Tämä tehtiin ajamalla aluksi Jenkinsin taustapalvelu alas ja poistamalla lopuksi kyseinen virtuaalikone Microsoft Azuresta.

7 OMAA POHDINTAA

Tämän opinnäytetyön toteutus oli aikataulullisesti kunnianhimoinen ja opettavainen projekti minulle. Ehkäpä haastavimpana asiana opinnäytetyön toteutusosaa tehdessä oli erilaisten uusien ohjelmistojen käytön opettelu, erilaisten verkotuskonfigurointien määrittäminen sekä lopullisen kokonaisuuden ymmärtäminen. Asennustyö tehtiin

myös suurimmilta osin Linuxin komentoriviltä, joten suurin osa Linuxin peruskomennoistakin tuli opetella ja muistaa, jotta toimintaympäristöt saatiin toimivaksi.

Toteutuksen dokumentoinnissa yritin pitää asennuksessa esiintyneet asiat mahdollisimman ymmärrettävänä ja suoraviivaisena myös sellaiselle lukijalle, jolle termit ja käytänteet eivät ole ennestään tuttuja. Tämän lisäksi myös jokaista yksityiskohtaa määrittelytiedostojen luomisessa ei ole kerrottu liian tarkasti, jotta opinnäytetyö ei olisi kasvanut sivumäärällisesti liian pitkäksi.

Jenkinsiin olin tutustunut aikaisemmin harjoitteluni aikana Codecontrol Oy:llä, mutta en kuitenkaan tarkemmin sen syvempiin ominaisuuksiin ja määrityksiin. Eri teorialähteet, Internetin hakutulokset sekä saatu apu kuitenkin tukivat työn valmistumista ja madalsivat oppimiskynnystä toteutuksen aikana.

Ansibleen sain ensikosketuksen toteutuksen aikana. Sen toimintaperiaate asennuksen kokonaisvaltaisesta automatisoinnista kuitenkin alkoi avautua vasta sitä mukaa kun käsittelin ja rakensin rooleja ja Playbook-tiedostoja sekä ratkoin asennuksissa esiintyneitä virheitä. Virhetilanteita asennuksissa aiheuttivat eniten slave-järjestelmiä varten itse luotu rooli, jonka sisältöä piti korjata ja testata useasti ennen kuin asennus saatiin loogisesti toimivaksi ja slave-järjestelmät alkoivat toimia odotetusti master-järjestelmässä. Suunnitteluvaiheessa Ansiblen hierarkkisuuden rakentaminen ja sen ymmärtäminen olivat myös tärkeässä osassa.

Loppujen lopuksi omasta mielestäni toteutus onnistui ilman suurempia ongelmia ja opinnäytetyö aikataulullisesti myös hyvin. Näin jälkikäteen voin todeta, että teoriaosuuden kokoaminen antoi hyvät pohjatiedot ketterää ohjelmakehitystoimintamallia tukevista käytännöistä ja toteutustyö erittäin hyvät taidot tehdä vastaavanlaisia projekteja Ansiblella myös tulevaisuudessa. Tästä on hyvä jatkaa.

LÄHTEET

Ansible a. 2018. About Ansible. Viitattu 24.3.2018. Saatavissa: http://docs.ansible.com/ansible/latest/index.html

Ansible b. 2018. Roles. Viitattu 28.3.2018. Saatavissa: http://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

Duvall P., Matyas S., Glover, A. 2007. Continuous integration: improving software-quality and reducing risk. Addison-Wesley Professional.

Fowler, M. 2006. Continuous Integration. Viitattu 12.3.2018. Saatavissa: https://www.martinfowler.com/articles/continuousIntegration.html

Haikala I. & Mikkonen T. 2011. Ohjelmistotuotannon käytännöt, 12. uudistettu painos. Hämeenlinna. Talentum.

Humble, J. & Farley D. 2011. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston: Pearson Education, Inc.

Jenkins a. Hardware Recommendations. Viitattu 15.3.2018. Saatavissa: https://jenkins.io/doc/book/hardware-recommendations/

Jenkins b. LTS Release Line. Viitattu 17.3.2018. Saatavissa: https://jenkins.io/download/lts/

Jenkins c. Managing Plugins. Viitattu 20.3.2018. Saatavissa: https://jenkins.io/doc/book/managing/plugins/

Jenkins d. Pipeline. Viitattu 18.3.2018. Saatavissa: https://jenkins.io/doc/book/pipeline/

Jenkins e. Pipeline Syntax. Viitattu 20.3.2018. Saatavissa: https://jenkins.io/doc/book/pipeline/syntax/

Jenkins f. Getting started with Pipeline. Viitattu 20.3.2018. Saatavissa https://jenkins.io/doc/book/pipeline/getting-started/

Jenkins g. Say Hello to the Blue Ocean Pipeline Editor. Viitattu 20.3.2018. Saatavissa: https://jenkins.io/blog/2017/02/15/pipeline-editor-preview/

Kasurinen, JP. 2013. Ohjelmistotestauksen käsikirja, 1. painos. Jyväskylä. Docendo Oy.

Mukherjee, Juni. 2018. Continuous Delivery 101. Viitattu 11.4.2018. Saatavissa: https://www.atlassian.com/continuous-delivery/pipeline

Smart, J. 2011. Jenkins: The Definite Guide. Sebastopol: O'Reilly Media Inc. Viitattu 12.3.2018. Saatavissa: http://barbra-coco.dyndns.org/student/Jenkins-%20The%20Definitive%20Guide.pdf

Tutorialspoint. 2018. Extreme Programming – Practices. Viitattu 12.3.2018. Saatavissa: https://www.tutorialspoint.com/extreme_programming/extreme_programming_practices.htm

Visual Studio Code. 2018. Getting Started. Viitattu 4.4.2018. Saatavissa https://code.visualstudio.com/docs

VirtualBox. Chapter 1. First Steps. Viitattu 24.3.2018. Saatavissa: https://www.virtualbox.org/wiki/End-user_documentation