Vaasan yliopisto
UNIVERSITY OF VAASA

Vesa Ranta-Kuivila

Jatkuvan integraation ja jatkuvan toimittamisen toimintamallin hyödyntäminen tietoverkon hallinnassa

Tekniikan ja innovaatiojohtamisen akateeminen yksikkö
Kauppatieteiden maisteri, pro gradu -tutkielma
Tietojärjestelmätieteen maisteriohjelma

Vaasa 2020

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen akateeminen yksikkö

Tekijä:	Vesa Ranta-Kuivila
Tutkielman nimi:	Jatkuvan integraation ja jatkuvan toimittamisen toimintamallin hyödyntäminen tietoverkon hallinnassa
Tutkinto:	Kauppatieteiden maisteri
Oppiaine:	Tietojärjestelmätieteen maisteriohjelma
Työn ohjaaja:	Tero Vartiainen
Valmistumisvuosi:	2020 **Sivumäärä:** 85

TIIVISTELMÄ:

Jatkuvan integrointi ja jatkuva toimittaminen on toimintamalli, jota on hyödynnetty sovelluskehityksessä sekä sovellusympäristöjen ylläpidossa. Se on osaltaan mahdollistanut sellaisten tietojärjestelmien kehityksen, joissa tänä päivänä pyritään automatisoimaan toistuvien tehtävien suorittamista, ja jotka ovat käytettävissä paikasta riippumatta. Tietoverkot ovat osa näitä tietojärjestelmiä, ja niiden ylläpidolta vaaditaan vastaavanlaista ketteryyttä, varmuutta ja täsmällisyyttä. Yhtenä mahdollisuutena kehittää tietoverkkojen hallintaa nähdään jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin hyödyntäminen. Tässä tutkimuksessa selvitetään millainen tietoverkon hallinnan toimintamalli voisi olla. Verkonhallinnan näkökulmasta mm. erilaiset verkkoympäristöjen valvontatyökalut ovat osa hallintakokonaisuutta, ja myös niissä on omat mahdollisuutensa tehdä ohjelmallisia konfiguraatioita automaattisesti. Tutkimuksessa toimintamallin hyödyntäminen on kuitenkin rajattu verkkolaitteiden konfiguraatioiden tekemiseen.

Tutkimus suoritettiin käyttämällä suunnittelutieteellistä tutkimusmenetelmää. Tutkimusongelma asetettiin alfa-yrityksessä muodostuneen tarpeen mukaisesti. Suunnittelusykli aloitettiin kokoamalla jatkuvan integroinnin ja jatkuvan toimittamisen tutkimustiedosta vaatimukset, jotka toimintamallin sovitus verkonhallintaan tulisi teorian pohjalta täyttää. Sykli eteni toimintamallin rakentamisella, ja arviointivaiheessa havainnoimalla miten artefakti pystyi vastaamaan asetettuihin vaatimuksiin. Lopuksi artefakti esiteltiin tutkimusongelman asettaneelle organisaatiolle simuloimalla. Organisaation palautteen lisäksi havainnoitiin, miten artefakti vastasi alkuperäiseen asetettuun ongelmaan.

Jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia hyödyntämällä on mahdollista toteuttaa käyttöönottoputki, jolla verkkolaitteiden konfiguraatioita voidaan kehittää, testata, provisioida ja todentaa toimiviksi. Verrattuna manuaalisesti tehtävään provisointiin, konfiguraatioiden ajaminen verkkolaitteisiin tapahtuu toimintamallin avulla täsmällisesti sekä useisiin laitteisiin yhtäaikaisesti skaalautuen. Provisioinnin jälkeinen automaattinen, konfiguroitujen palveluiden toiminnallisuuden testaus on oleellinen mallin tuoma etu. Tärkeää on konfiguraatioita määritellessä suunnitella myös niitä vastaava testaus-skripti. Kun muutos tehdään, kaikki testit ajetaan automaattisesti. Tämä on huomattava parannus manuaalisesti tehtäville ja silmämääräisesti tarkistettaville testeille.

AVAINSANAT: Integrointi, toimittaminen, tietoverkot, toimintamalli

Sisällys

1 Johdanto 7

1.1 Tutkimusmetodi 9

1.2 Tutkimuskysymys 9

1.3 Tulokset 10

1.4 Tutkimusraportin rakenne 11

2 Jatkuva integrointi ja jatkuva toimittaminen 12

2.1 Versionhallinta 13

2.2 Jatkuva integrointi 15

2.2.1 Nopea palaute 15

2.2.2 Versionhallinta jatkuvassa integroinnissa 16

2.2.3 Testaus 17

2.2.4 Ohjenuorat työnkulun suunnitteluun 18

2.3 Jatkuva toimittaminen 19

2.3.1 Nopea palaute ja lyhyempi kierrosaika 20

2.3.2 Käyttöönottoputki 20

2.3.3 Käyttöönottoputken mallinnus 22

2.3.4 Käyttöönottoputken toteuttaminen 24

2.4 Konfiguraation hallinta 26

3 Tutkimusmenetelmä 30

3.1 Suunnittelutieteellinen tutkimus 30

3.2 Suunnittelutieteellisen tutkimuksen viitekehys 31

3.2.1 Viitekehyksen elementit 31

3.2.2 Viitekehyksen syklit 32

3.3 Suositukset suunnittelutieteellisen tutkimustyön tekemiselle 34

3.4 Suunnittelutieteellisen tutkimusmenetelmän prosessi 35

4 Toimintamallin tutkimusprosessi 38

4.1 Ongelman tunnistaminen ja motivointi 38

4.2 Artefaktin tavoitteet 39

4.3 Suunnittelu ja kehitys 40

4.3.1 Jatkuvan integroinnin ja jatkuvan toimittamisen työkalut 42
4.3.2 Konfiguraationhallintatyökalu 43
4.3.3 Verkkoympäristön virtualisointi 44
4.3.4 Tutkimusympäristö 45
4.4 Käyttöönottoputken havainnollistaminen 47
5 Arviointi 56
5.1 Nopea palaute 58
5.1.1 Palaute testauksesta CI:ssä ja CD:ssä 58
5.1.2 Jatkuvalla toimittamisella saavutettava nopea kierrosaika ja asiakaspalaute 59
5.1.3 Uusilla palveluilla oltava lyhyet kehityssyklit 60
5.1.4 Työnkulun on oltava välitön 60
5.1.5 Työnkulun on oltava läpinäkyvä 61
5.1.6 Tiimin yhteistyöllä rakentama käyttöönottoputki 61
5.2 Versionhallinta 62
5.2.1 Käytä vain päälinjaa 62
5.2.2 Hyödynnä kommunikointivälineenä 62
5.2.3 Päälinjan on pysyttävä eheänä 63
5.3 Testaus 63
5.3.1 Suoritusaika voi olla muutamia minuutteja 63
5.3.2 Yksikkötestit 65
5.3.3 Komponenttitestit 66
5.3.4 Hyväksyntätestit 66
5.4 Palautus 67
5.4.1 Epäonnistunut toimitus tulee pystyä palauttamaan 67
5.4.2 Palautuksen toimivuuden testaus säännöllisin väliajoin 67
5.5 Konfiguraationhallinta 67
5.5.1 Yleiskäyttöiset skriptit siten, että samoilla skripteillä toimitetaan sekä testi- että tuotantoympäristöön 67

5.5.2 Kaksivaiheisella testauksella varmistetaan laitteiden tavoitettavuus ja palveluiden oikeanlainen toiminta. 68

6 Diskussio 69

6.1 Tulokset 70

6.2 Johtopäätökset 73

6.3 Rajoitukset 74

Lähteet 75

Liitteet 79

Liite 1. Käyttöönottoputken .gitlab-ci.yml -konfiguraatio 79

Kuvat

Kuva 1. Versiohaarojen käyttö ohjelmistoprojektissa. 15
Kuva 2. Käyttöönottoputki sekä sen vaikutukset sovelluksen toimittamiseen ja kehittämiseen. 21
Kuva 3. Jatkuvan integroinnin työnkulun meta-malli. 23
Kuva 4. Suunnittelutieteellisen tutkimuksen viitekehys. 31
Kuva 5. DSRM prosessimalli. 36
Kuva 6. Tutkimuksessa käytetty testiverkko. 45
Kuva 7. Tutkimusympäristön komponentit ja niiden väliset riippuvuudet. 46
Kuva 8. Tutkimuksessa toteutettu käyttöönottoputki. 48
Kuva 9. Käyttöönottoputken aktiviteettien ohjaus eri palvelimille kohdeympäristöstä riippuen. 49
Kuva 10. Käyttöönottoputkessa suoritettavat aktiviteetit, kun kommitoidaan kehityshaaraan. 50

Taulukot

Taulukko 1. Suunnittelutieteellisen tutkimuksen suositukset. 34
Taulukko 2. Rakentamisvaiheessa käyttöönottoputkelle asetetut tavoitteet. 41
Taulukko 3. Käyttöönottoputken solmut, tyypit ja kuvaukset. 53
Taulukko 4. Käyttöönottoputkelle testatut suoritusajat. 64

1 Johdanto

Osana yrityksen tietojärjestelmää tietoverkot, ja niiden hallinta, ovat uusien vaatimusten äärellä. Yksi syyllinen siihen on kehitys, jonka konesalien palvelimet ovat viime vuosien aikana käyneet läpi. Fyysisistä yhden raudan palvelimista on siirrytty helposti käyttöönotettaviin ja monistettaviin virtuaalipalvelimiin. Lisäksi konesali on siirtynyt yritysten omista tiloista vuokrattavaksi pilvipalveluksi, jossa palveluntarjoaja huolehtii fyysisen kapasiteetin riittävyydestä, kuten Amazonin AWS ja Microsoftin Azure. Yritys voi siten ketterästi hyödyntää sekä omia että vuokrattavia resursseja kulloisenkin tarpeensa mukaisesti. Helpon käyttöönoton, konfiguroinnin ja lopulta käytöstä poistamisen on tehnyt mahdolliseksi automatisointia hyödyntävät prosessit, joissa apuna on käytetty konfiguraationhallintatyökaluja, kuten Ansible ja Puppet (Ansible, 2016; Puppet, 2019). Näillä työkaluilla käyttöönotto on nopeaa ja täsmällistä, kun samalla vältetään manuaalisesta konfiguroinnista helposti aiheutuvat virheet tai puuttuvat konfiguraatiot.

Tietoverkoissa tämä kehitys on näkynyt kasvaneiden tiedonsiirtokapasiteettitarpeiden täyttämisenä. Lisääntynyttä kapasiteettia on tarvittu sekä konesaleissa palvelimien väliseen liikennöintiin että verkon reunalle yrityksien ja kotitalouksien tarpeisiin, kuten Cisco Systemsin (2018, 2019a) ennuste- ja trendiraporteista käy ilmi. Sovelluksia ei enää asenneta omalle koneelle, vaan niitä ajetaan joko yrityksen omasta konesalista tai palveluna internetin yli, jolloin nopeiden tietoliikenneyhteyksien merkitys korostuu. Toisaalta verkkolaitteiden hallintamenetelmät eivät ole kehittyneet palvelinresurssien hallinnan tahdissa. Kun palvelimen, sen käyttöjärjestelmän, ja siinä ajettavan sovelluksen voi käynnistää konfiguraationhallintatyökalujen avulla minuuteissa, muutokset tietoverkkoihin konfiguroidaan manuaalisesti. Yrityksen tietojärjestelmässä tietoverkkojen hallinnoinnin reagointi ja toimintakyky on siten automatisoinnin puutteesta johtuen heikointa (Cisco Systems, 2019b). Toisaalta virtuaalialustalla toimivalla sovelluksella ei ole samanlaista riippuvuutta fyysiseen laitteeseen, kuin tietoverkossa toteutetuilla palveluilla. Se on helppo sammuttaa, siirtää ja käynnistää uudelleen toisella virtuaalipalvelimella täysin ohjelmallisesti. Myös yksittäisen sovelluksen palvelukatkos, sovelluksesta tietenkin riippuen, ei välttämättä ole yrityksessä kriittinen, vaikka katkos vaikuttaisi kaikkiin sen

käyttäjiin. Tietoliikenneverkossa katkos sen sijaan on kokonaisvaltaisempi, koska se saattaa estää kerralla useamman sovelluksen käyttämisen. Siten sovellusympäristön konfigurointi on joustavampaa ja se ei ole niin haavoittuvainen virheille.

Työn ja sovellusten joustavuus tuovat lisää vaatimuksia tietoverkkojen konfiguroinnille. Työntekijöillä on enenevissä määrin vapaus valita työnsä suorituspaikka, kunhan käytettävissä on pääsy yrityksen tietoverkkoon ja sitä kautta sen sisäisiin tietojärjestelmiin. Sovellukset voivat sijaita yrityksen palvelimilla niiden omissa konesaleissa, pilvipalvelussa, tai ne voivat olla toteutettuna mikropalveluina, jolloin ne voivat sijaita kummassa ympäristössä tahansa. Verkon provisiointi, eli tietoliikennepalvelun konfiguraation tekeminen verkkolaitteisiin siten, että käyttäjällä on pääsy siihen ympäristöön, jossa hänen käyttämänsä sovellus sijaitsee, pitää kyetä tekemään nopeasti ja virheettömästi. Jotta tietoverkkoon ei tällaisissa ympäristöissä muodostu konfiguraatioita, jotka sallivat asiattomien pääsyn verkon sisäpuolelle, on täsmällisten konfiguraatioiden tekeminen tärkeää. Siinä merkittävä vaikutus on konfiguraatioiden automatisoinnilla. (Cisco Systems, 2019b.)

Verkkolaitteiden hallintatyökaluissa ja -metodeissa muutos on kuitenkin ollut hidasta. Provisiointi tapahtuu useimmiten komentoriviltä tai selainpohjaisen graafisen käyttöliittymän avulla. Komentorivi on lisäksi aina valmistajakohtainen käyttöliittymä. Komentojen termit ja syntaksi, eli missä järjestyksessä komentojen argumentit syötetään, vaihtelevat laitevalmistajien ja jopa valmistajan omien laitesarjojen välillä. Graafisesta käyttöliittymästä oikeat komennot voi olla helpompi löytää, mutta käytettävyydeltään se on hitaampi kuin komentorivi. Kummassakin tapauksessa konfigurointi tapahtuu manuaalisesti, ja samaan lopputulokseen voi päätyä useamman erillisen ja eri järjestyksessä suoritetun vaiheen kautta. Menetelmät ovat siten virhealttiita, ja koska niillä tehdään muutoksia tuotannossa oleviin laitteisiin, voidaan konfiguraatiolla aiheuttaa helposti tietojärjestelmään palvelukatkos.

Verkkolaitteiden valmistajat ovat alkaneet kehittää ja markkinoida menetelmiä, joilla verkonhallinnan tehokkuutta voitaisiin kasvattaa. Laitteiden ohjelmistoihin on lisätty tuki

API-rajapinnoille (engl. Application Programming Interface), joiden avulla konfiguraatio voidaan tehdä ohjelmallisesti. Jotta rajapintoja pystytään ylipäätään hyödyntämään, täytyy verkonhallinnassa työskentelevien omaksua uusia taitoja ja menetelmiä. Ohjelmointiosaaminen, automaattisesti tehdyt konfiguraatiot sekä konfiguraationhallintajärjestelmien käyttöönotto nähdään verkkolaitevalmistajien puolelta oleellisina työkaluina ja toimintatapoina (Cisco Systems, 2019b; Juniper Networks, 2019). Niiden avulla provisioinnin tehokkuutta saadaan parannettua. Lisäksi se tulee tapahtua luotettavasti, ilman virheellisistä konfiguraatioista aiheutuvia palvelukatkoksia. Jatkuvan integroinnin, CI:n (engl. Continuous Integration) pienissä paloissa tehty kehitystyö, ja jatkuvan toimittamisen, CD:n (engl. Continuous Delivery) automatisoitu testaus, yhdistettynä CI/CD menetelmäksi, mahdollistaa tällaisen toimintamallin luomisen konfiguraation hallinnalle ja provisioinnin automatisoinnille (Clark, 2018; Juniper Networks, 2019).

1.1 Tutkimusmetodi

Tutkimus tehdään käyttäen suunnittelutieteellistä tutkimusmenetelmää. Alfa-yrityksessä on havainnoitu muutos, joka verkonhallintamenetelmissä on tapahtumassa, ja jota tietoliikenneverkkolaitteiden toimittajat ovat osaltaan ajamassa markkinoille. Koska muutoksen tuomat menetelmät ovat liiketoiminnalle uusia, sopii suunnittelutieteellinen tutkimus hyvin tunnistetun ongelman ratkaisemiseksi. Lisäksi alfa-yritys voi hyödyntää tutkimuksen tuloksena syntyvää artefaktia omassa toiminnassaan.

1.2 Tutkimuskysymys

Tietoverkot ovat osa yritysten tietojärjestelmiä, ja siinä kokonaisuudessa tarve verkkojen konfiguroinnin automatisoinnille on kasvamassa. Konfiguraatiot tulee tehdä samalla tavalla tehokkaasti ja täsmällisesti kuin palvelimien ja sovellusten provisiointi. Verkkolaitevalmistajat ovat jo aiemmin reagoineet sovellusten hallinnassa tapahtuneeseen muutokseen lisäämällä ohjelmoitavia rajapintoja tietoliikennelaitteidensa käyttöjärjestelmiin.

Niiden hyödyntäminen on kuitenkin jäänyt vähäiseksi, ja verkonhallinnan siirtymä automatisaatioon ei ole edennyt. Ratkaisuksi tietoliikennealalla ollaan ehdottamassa samojen, hyväksi havaittujen menetelmien hyödyntämistä, kuin mitä konesalissa on palvelimien ja sovellusten osalta käytetty. Yksi menetelmistä on jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin hyödyntäminen siten, että automatisoidut provisioinnit voidaan tehdä luotettavasti tuotantokäytössä oleviin verkkolaitteisiin.

Alfa-yritys tuottaa verkonhallintaa jatkuvana palveluna. Siihen sisältyvät mm. verkon suunnittelu, rakentaminen, muutoshallinta, monitorointi sekä häiriöihin reagoiminen. Hallittavien verkkojen toteutukset räätälöidään asiakkaiden tarpeiden mukaan, ja niitä tehdään mm. lähiverkkoihin ja konesaleihin. Lisäksi käytössä on useita laitevalmistajia ja tekniikoita, esim. reitittimet, kytkimet, langattomat verkkolaitteet ja palomuurit. Koska CI/CD-menetelmää on käytetty sovellusten ja niihin liittyvien palvelinympäristöjen kehityksessä ja operoinnissa, on menetelmä alfa-yrityksen näkökulmasta vieras. Tutkimuskysymyksenä on siten: Miten hyödyntää jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia verkkolaitteiden hallintaan lähiverkon palvelutuotannossa?

1.3 Tulokset

Raportin tulos osoittaa, että jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia hyödyntämällä on mahdollista toteuttaa käyttöönottoputki, jolla verkkolaitteiden konfiguraatioita voidaan kehittää, testata, provisioida ja todentaa toimiviksi. Toimintamalliin liittyvät testaukset sovelluskehityksen eri vaiheissa ovat olennainen osa mallia myös verkon konfiguraatiossa, ja automaattisesti suoritettuina ne mahdollistavat huomattavan parannuksen silmämääräisesti tehdyille muutosten tarkistuksille.

1.4 Tutkimusraportin rakenne

Raportti on jaettu kuuteen kappaleeseen. Johdanto kappaletta seuraa teoriaosuus, jossa käsittelen jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia kirjallisuuskatsauksena. Kappaleessa käydään lyhyesti läpi myös versionhallinta, joka on kiinteä osa mallia. Tutkimusmenetelmän ja menetelmään sovitetun tutkimusprosessin esittelen kappaleessa kolme. Tutkimuksen suorittamisen, tutkimusympäristön sekä tutkimuksen tuloksena syntyneen toimintamallin mukaisen käyttöönottoputken kuvaan kappaleessa neljä. Kappaleessa viisi esittelen tulokset toimintamallin suunnittelussa syntyneistä havainnoista, sekä alfa-yrityksen arvion toimintamallin soveltuvuudesta yrityksen ympäristössä. Kappaleessa kuusi esittelen tutkimuksen johtopäätökset ja rajoitukset.

2 Jatkuva integrointi ja jatkuva toimittaminen

Kun ohjelmistoa kehitetään usean kehittäjän toimesta osissa, jossain vaiheessa ohjelmistoprojektia tulee hetki, jolloin osat integroidaan eli yhdistetään toimivaksi sovellukseksi. Mitä pidempään erillistä kehitystyötä tehdään, sitä vaativammaksi integrointi muodostuu. Kehittäjät tekevät itsenäisiä ratkaisuja ohjelmoidessaan omaa osuuttaan. Integraatiossa sovelluksen toimivuus testataan, jolloin selviää miten nämä ratkaisut toimivat yhdistettynä toisiinsa. Helposti käy niin, että integraatiossa joudutaan ratkomaan yhteensovittamisongelmia, ja suunnittelemaan sekä ohjelmoimaan osia uudelleen. Jatkuva integrointi pilkkoo pitkät kehitysjaksot mahdollisimman lyhyiksi, jotta massiivisilta yhteensovittamisilta vältytään. Samalla koko sovelluksen testaus aloitetaan aikaisessa vaiheessa projektia. (Fowler, 2006.)

Manuaalisessa sovelluksen toimituksessa julkaisu asiakkaan tuotantoympäristöön voi olla projektitiimille stressaava toimenpide. Se on kertaluontoinen tapahtuma, jolloin sen identtinen toistaminen on hyvin vaikeaa. Manuaalisesti tehtävät konfiguraatiot ovat virhealttiita, ja tehtyjen toimenpiteiden jäljittäminen on työlästä. Etenkin jos sovelluksen toiminnassa on havaittu virheitä, ja on jouduttu tekemään nopeita korjauksia joko itse sovellukseen tai sen konfiguraatiotietoihin. Tällöin korjausten dokumentointi voi jäädä tekemättä kokonaan. Asiaa ei helpota se, että toimitukseen osallistuu yleensä useita tiimejä, joiden toiminnan koordinointi etenkin ongelmatilanteissa on haasteellista. Jokaisen toimituksen lopuksi sovelluksen toimivuus todennetaan vielä manuaalisesti tehdyillä testeillä, jolloin tapahtuman kertaluontoisuus jälleen korostuu. Jatkuva toimittaminen pyrkii ratkaisemaan yllä mainittuja ongelmia runsaalla testauksella ja toimintojen automatisoinnilla. Sovelluksen toimittaminen, konfigurointi ja testaus tehdään kaikki valmiiksi määritellyillä komento-skripteillä, jotka samalla toimivat dokumentaationa julkaisussa tehdyille toimenpiteille. Skriptit voidaan ajaa useaan kertaan, ja ne toteuttavat julkaisun aina samalla tavalla. (Humble & Farley, 2010, s. 4–7.)

Tässä luvussa käyn läpi jatkuvan integroinnin sekä jatkuvan toimittamisen toimintamalleja. Ne yhdistämällä on mahdollista luoda työnkulku, jossa kehitys-, testaus- ja

toimitusvaiheet ovat sovitettu yhteen siten, että työnkulun läpivienti tapahtuu pääasiassa automatisoitujen tehtävien avulla (Humble & Farley, 2010, s. 9–11). Toimintamallissa nämä vaiheet käydään projektin aikana läpi useaan kertaan, jolloin lopulta tuotantoon julkaisussa voidaan olla varmoja, että sovellus ja sen asennuskonfiguraatiot ovat toimivia ja että julkaisussa ei ilmene uusia ongelmia. Ennen perehtymistä jatkuvaan integrointiin ja jatkuvaan toimittamiseen, käyn erillisessä luvussa läpi versionhallinnan taustoja, koska se on hyvin olennainen osa jatkuvan integroinnin toimintamallia.

2.1 Versionhallinta

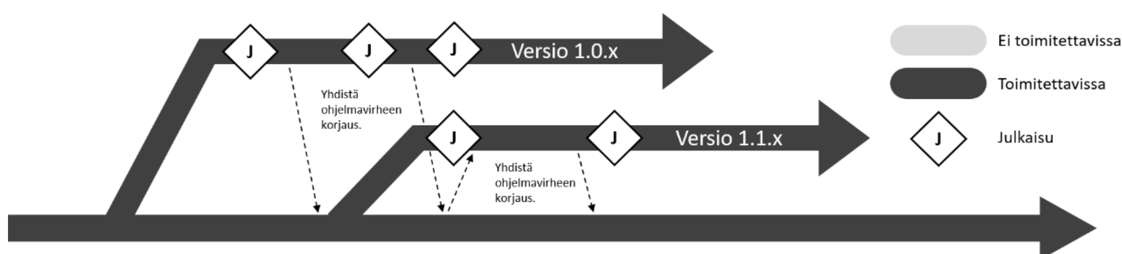
Versionhallintajärjestelmä pitää kirjaa tiedostoihin tehdyistä muutoksista. Tästä kirjanpidosta syntynyt muutoshistoria mahdollistaa eri ajanhetkinä tallennettujen tiedostoversioiden, ja tarkemmin niiden sisältöjen, vertaamisen keskenään. Etenkin ohjelmistoprojekteissa, joissa työskennellään tekstipohjaisilla tiedostoilla. Vanhoihin muutoksiin on siten yksinkertaista palata, ja niitä on myös helppo palauttaa, jos poistettu ohjelman osa halutaankin hyödyntää myöhemmin. Toisaalta muutoshistoria kertoo tarinaa miten ohjelmiston kehitys on edennyt. Sovelluksen kehittäjän näkökulmasta kehitystilanteen taltioinnin lisäksi versionhallinta toimii myös yhteistyötyökaluna muiden kehittäjien kanssa. Versiointi pitää kirjaa kuka teki minkäkin muutoksen ja milloin. Yhteydenotto toiseen kehittäjään helpottuu, kun tiedon saa nopeasti esiin versionhallinnasta. Myös yhtäaikainen kehitystyö ohjelman eri osissa on mahdollista, ja versionhallinta säilöö aina viimeisimmän tiedon koko ohjelmiston sisällöstä. (Chacon & Straub, 2014/2020; Humble & Farley, 2010.)

Yksinkertaisimmillaan versionhallinta voi toimia käyttäjän omalla koneella. Se ei kuitenkaan mahdollista sujuvaa yhteistyötä muiden käyttäjien kanssa. Tähän tarkoitukseen onkin olemassa kahdenlaisia toteutuksia: keskitettyjä sekä jaettuja versionhallintajärjestelmiä. Keskitetty versionhallinta tapahtuu palvelimella, josta kehittäjät kuittaavat (engl. checkout) tiedostot omalle tietokoneelleen työstettäväksi, ja johon he kommitoivat (engl. commit) valmiit muutokset. Tässä yhteydessä kehittäjä kuvailee lyhyesti, mitä muutoksia

kyseinen tallennus sisältää, ja samalla muutoshistoriaan tallentuu, kuka muutoksen on tehnyt ja milloin. Versionhallintajärjestelmä ottaa päivitykset vastaan, ja yhdistää ne aikaisempaan versioon luoden uuden version muuttuneista tiedostoista. Samalla palvelin on kuitenkin yksittäinen vikaantumispiste. Jaettu versionhallintajärjestelmä toimii tiedostojen versionhallinnan osalta samalla periaatteella, mutta siinä ei ole yhtä keskitettyä palvelinta, vaan asiakaskoneelle kopioidaan koko versionhallinta historiatietoineen. Näin mikä tahansa kone, joka on liitetty jaettuun järjestelmään, voi toimia palvelimena. Kommitointi tehdäänkin siten aina kehittäjän omalla koneella, josta muut käyttäjät voivat ladata muutokset ja yhdistää omalla koneella olevaan versioon. (Chacon & Straub, 2014/2020; Humble & Farley, 2010.) Jaetussa versionhallinnassa käytetään kuitenkin usein keskitettyä palvelinta, josta kehittäjien on helppo ladata muiden tekemät muutokset omalle koneelleen, kuin myös päivittää omat muutoksensa muiden saataville. Keskitetyn palvelimen käyttö mahdollistaa myös jatkuvan integroinnin toimintamallin hyödyntämisen (Humble & Farley, 2010, s. 393).

Kun uuden projektin kehitystyö aloitetaan, sijaitsevat tiedostot versionhallinnan päälinjassa. Versionhallinnan yksi olennaisista ominaisuuksista on mahdollisuus luoda versiosta haaroja (engl. branches), joissa kehitystä voidaan tehdä rinnakkain esimerkiksi siten, että kukin kehittäjä keskittyy työhön omassa haarassaan vaikuttamatta muiden työhön. Lähtötilanne, josta haara luodaan, voi olla joko päälinja tai aiemmin luotu haara. Kehitystyö tulisi pyrkiä määrittelemään siten, että kehittäjien tekemät muutokset osuvat keskenään mahdollisimman vähän päällekkäin. Haarassa tehty valmis työ yhdistetään lopulta takaisin esim. päälinjaan, ja tällöin päällekkäisistä muutoksista muodostuu yhdistämisongelmia. Näissä tapauksissa yhdistämisen tekevä kehittäjä ratkaisee, mitkä päällekkäiset muutokset jäävät voimaan versionhallintaan. Riippuen tehtyjen muutosten määrästä näiden ongelmien selvitys voi olla hyvin työlästä. (Chacon & Straub, 2014/2020, s. 62–76; Humble & Farley, 2010, s. 388–389). Haarojen tekeminen on versionhallinnassa helppoa, ja niitä voidaan luoda eri perusteilla, kuten ohjelmistoversioilla, uuden toiminnon kehityksellä tai organisaation tiimirakenteilla. Humble ja Farley (2010) suosittelevat kuitenkin välttämään niiden käyttämistä, ja kehottavat tekemään kehitystyötä

pelkästään päälinjaan. Ainoastaan versiohaarat ovat heistä hyväksyttäviä, koska ne jatkavat olemassa oloaan erillisinä, korkeintaan ohjelmavirheiden korjauspäivityksiä sisältävinä haaroina. Tätä on havainnollistettu alla olevassa kuvassa. Alimpana olevasta päälinjasta on tehty kaksi versiohaaraa. Ohjelmavirheiden korjausten myötä niistä on ainoastaan silloin julkaistu päivitetty versio. Varsinainen kehitystyö jatkuu päälinjassa, johon myös versiohaaroissa tehdyt ohjelmavirheiden korjaukset yhdistetään.



Kuva 1. Versiohaarojen käyttö ohjelmistoprojektissa (mukaillen Humble & Farley, 2010, s. 392).

2.2 Jatkuva integrointi

2.2.1 Nopea palaute

Fowlerin (2006) mukaan jatkuvalla integroinnilla voidaan välttää pitkien kehitysjaksojen aiheuttamia ongelmia sovelluskehityksessä. Ajatuksena on, että integrointia tehdään niin usein kuin mahdollista, sen sijaan että projektin lopussa tehtäisiin yksi iso integraatio. Jokainen kehittäjä integroi ohjelmakoodinsa vähintään kerran päivässä, mieluummin useammin. Sovellus rakentuu pala palalta, ja lisäksi sen toimivuus testataan jokaisen integraation yhteydessä. Monimutkaisia yhteensovittamisongelmia ei pääse muodostumaan, vaan alkavat integraatio-ongelmat saadaan kiinni aikaisessa vaiheessa, jolloin niihin voidaan reagoida nopeasti. Nopea palaute onkin jatkuvan integroinnin yksi keskeisistä hyödyistä.

2.2.2 Versionhallinta jatkuvassa integroinnissa

Versionhallinta on oleellinen osa jatkuvaa integrointia. Kun kehittäjät kommitoivat ohjelmakoodia keskitetyn versionhallinnan päälinjaan mahdollisimman usein, pysyy päälinjan koodi tuoreena, jolloin jokaisella kehittäjällä on käytettävissään sovellukselle tehdyt viimeisimmät muutokset. Ennen kommitointia kehittäjän tehtävänä onkin ladata mahdolliset muutokset omaan kehitysympäristöönsä. Versionhallinnan työkalujen avulla varmistetaan, etteivät omat koodimuutokset aiheuta päälinjalla olevan koodin kanssa yhdistämiskonfliktia, eli tilannetta, jossa useampi kehittäjä on muokannut samaa kohtaa koodista. Mikäli näin käy, kehittäjän tulee tehdä korjaukset omaan versioonsa. Koska versionhallinnasta saa helposti tietoon kuka on tehnyt minkäkin muutoksen, voivat kehittäjät myös tehdä yhteistyötä kyseisen koodin osalta ja siten tuottaa kokonaisuutena paremmin toimivan toteutuksen sovellukseen. (Fowler, 2006; Humble & Farley, 2010). Versionhallinta on siten yksi nopean palautteen kanavista jatkuvassa integroinnissa. Se toimii myös kommunikaatiovälineenä, sisältäen tarkat tiedot päälinjaan tehdyistä muutoksista, kuka on tehnyt mitä ja milloin (Fowler, 2006; Humble & Farley, 2010, s. 76–77).

Strategia, miten versionhallintaa tulisi hyödyntää jatkuvassa integroinnissa, on hyvin yksinkertainen. Sekä Fowler (2006) että Humble ja Farley (2010) painottavat, että töitä tulisi tehdä ainoastaan päälinjassa. Vaikka haarojen tekeminen on versionhallinnassa helppoa, tulisi niitä heidän mukaansa käyttää ainoastaan sovelluksen versiointiin, tai kehityksen aikaiseen testaukseen, jonka päätyttyä haarauma poistetaan. Kun kehitystä tehdään omissa haaroissaan, muodostuu kehitysjaksoista helposti pidempiä, kuin jos toimittaisiin pelkästään päälinjassa. Tällöin integrointitiheys harvenee ja integroitavan koodin määrä kasvaa, mikä johtaa helpommin koodin yhteensovittamisongelmiin. Ståhl ja Bosch (2014a) vahvistavat strategian toimivuuden jatkuvaan integraatioon keskittyvän kirjallisuuden perusteella, ja muistuttavat miten sillä vältytään ”integraatiovelan” muodostumiselta. Sovelluksen kehittäminen erillisissä haaroissa on siten selvästi jatkuvan integroinnin toimintamallin vastaista.

2.2.3 Testaus

Jatkuvassa integroinnissa sovellus testataan aina, kun uutta koodia päivitetään versionhallinnan päälinjalle. Testauksella pyritään todentamaan, että uusi koodi ei itsessään sisällä virheitä, ja että se toimii myös osana koko sovellusta. Käytännössä ohjelmisto tulee tällä tavoin testattua lukemattomia kertoja projektin aikana. Testauksen ansiosta jatkuvan integroinnin oletus on, että päälinjalla oleva ohjelmisto on aina toimitettavissa tuotantoympäristöön. Tiheä testaaminen vaatii kuitenkin resursseja, ja siksi testaaminen tulee automatisoida. Testien käynnistäminen voi tapahtua joko manuaalisesti, tai ne voivat käynnistyä automaattisesti, kun kehittäjä kommitoi koodia päälinjaan. (Fowler, 2006; Humble & Farley, 2010.) Työnkulun ja sen tehtävien automatisointi nähdään luontaisena osana jatkuvan integraation toimintamallia (Ståhl & Bosch, 2014a).

Humble ja Farley (2010, s. 60–61) esittelevät integraatiovaiheen testaukseen kuuluvan yleensä kolmenlaisia testejä: yksikkö-, komponentti- ja hyväksyntätestejä. Yksikkötestit kehittäjä kirjoittaa testaamaan versionhallinnan päälinjaan tekemänsä ohjelmakoodin. Niillä testataan pieniä osia ohjelmasta, kuten yksittäisten funktioiden tai metodien toiminta. Komponenttitestit testaavat sovelluksen eri osien toimivuutta keskenään. Ne voivat olla myös yhteydessä ulkoisiin järjestelmiin, kuten tietokantoihin. Koska testiympäristössä ei kaikkia ulkoisia järjestelmiä ole välttämättä käytettävissä, ne on voitu korvata ohjelmallisilla tynkäversioilla, jotka imitoivat varsinaisen järjestelmän antamia vasteita. Hyväksyntätesteissä ajetaan koko ohjelmaa ja todennetaan, että ohjelmisto täyttää sille asiakkaan asettamat liiketoimintavaatimukset sekä toiminnallisesti että suorituskyvyltään. Ståhl ja Bosch (2014a, 53–54) toteavat myös, että määritelmä integraatiovaiheen testaussisällöstä vaihtelee. Minimissään se voi olla pelkästään yksikkötestausta.

Koska testejä tehdään usein, niiden suoritusaika ei saa kasvaa projektin kuluessa ja testattavan koodin määrän lisääntyessä liian pitkäksi. Muutoin testaajien kärsivällisyys jatkuvan integraation toimintamalliin saattaa joutua koetukselle. Pitkistä testeistä muodostuu myös huomattava resurssikustannus projektille. Fowler (2006) sekä Humble ja Farley (2010, s. 61) suosittelevat testauksen kestoksi maksimissaan kymmentä minuuttia.

Testausaika pitää sisällään yksikkötestauksen sekä testaajan omassa kehitysympäristössä että versionhallinnan päälinjaan kommitoidun koodin testauksen integrointiympäristössä. Jotta testaus tapahtuu tehokkaasti, se voidaan vaiheistaa siten, että nopeat testit ajetaan ensin ja hitaammat myöhemmin (Humble & Farley, 2010; Ståhl & Bosch, 2014a). Näin kehittäjä saa nopeasti palautteen kommitoimansa koodin toimivuudesta, ja voi siirtyä kehittämään seuraavaa osaa ohjelmasta. Esimerkiksi yksikkö- ja komponenttitestit antavat nopeasti palautteen, jolloin erilaiset hyväksyntätestit voivat jäädä testaustiimin ajettaviksi (Humble & Farley, 2010).

Jos jokin versionhallinnassa olevaa koodia vasten ajetuista testeistä epäonnistuu, jatkuvan integroinnin toimintamallissa se tarkoittaa että versionhallinnan päälinja on rikki. Tällöin muut kehittäjät eivät voi kommitoida omia töitään, koska ylimääräinen uusi koodi vain vaikeuttaisi virheen etsintää ja korjausta (Humble & Farley, 2010, s. 66). Päälinja tuleekin korjata mahdollisimman nopeasti etsimällä virhe viimeksi kommitoidusta koodista ja korjaamalla se, tai palauttamalla päälinja edelliseen toimivaan versioon (Fowler, 2006; Humble & Farley, 2010). Ståhl ja Bosch (2014a) sen sijaan havaitsivat alan kirjallisuudessa mielipiteiden vaihtelevan tässä asiassa laidasta laitaan. Vaativimmillaan kommitointitestauksen lisäksi tuli tehdä koodin katselmointi yhdessä toisen kehittäjän kanssa, kun keveimmillään koko testaus voitiin jättää tekemättä. Yleisesti voisi todeta, että virheiden tekemistä ei kannata vältellä liiaksi. Jatkuvassa integroinnissa on kuitenkin tärkeää, että kommitointia tehdään usein. Tiheällä syklillä tehdyissä päivityksissä uuden koodin määrä on pieni, jolloin virhe on helppo rajata, ja nopean testipalautteen ansiosta koodi on kehittäjällä vielä tuoreessa muistissa (Fowler, 2006).

2.2.4 Ohjenuorat työnkulun suunnitteluun

Ståhl ja Bosch (2014b, s.61–62) tutkivat, miten jatkuvan integroinnin menetelmiä oli käytännössä hyödynnetty erilaisten teollisuusyritysten sovelluskehitystiimeissä. Tutkimustulosten perusteella he kokosivat seuraavat ohjenuorat jatkuvan integroinnin työnkulkujen muodostamiselle:

1. Kokonaisvaltaiset testit. Automatisoidut testit tulisi rakentaa niin laajoiksi, että ne antavat riittävän varmuuden ohjelman toimivuudesta. Liian yleinen testaus luo virheellisen turvallisuuden tunteen.
1. Tehokas kommunikaatio. Kehittäjät tarvitsevat palautteen testien onnistumisesta, ja heillä tulee olla pääsy ja ymmärrys automatisoidun testauksen työnkulkuun.
2. Välittömyys. Kehittäjän tulee pystyä kommitoimaan koodinsa testauksen työnkulkuun aina halutessaan.
3. Työnkulun mitoitus. Testausympäristön kapasiteetti tulee mitoittaa niin, että se pystyy käsittelemään sisään tulevat testit riittävällä nopeudella.
4. Tarkkuus. Testauksen työnkulussa lähdekoodista käännetty ajettava ohjelma pitää olla käytettävissä myöhemmissä testivaiheissa. Työnkulkua ei tule suunnitella niin, että lähdekoodin kääntäminen tehdään eri vaiheissa uudelleen, ja mahdollisesti eri järjestelmäympäristössä. Tällöin ohjelmaversiot eivät välttämättä ole identtisiä keskenään, jolloin eri vaiheiden testit eivät kohdistu enää alkuperäiseen ohjelmaan.
5. Läpinäkyvyys. Työnkulusta tulee tehdä selkeä ja yksikäsitteinen, jotta se on helposti kaikkien osallisten ymmärrettävissä.

2.3 Jatkuva toimittaminen

Jatkuvassa integroinnissa sovellusta kehitetään siten, että se on aina toimitettavassa tilassa. Jatkuva toimittaminen on integroinnille seuraava vaihe. Se on toimintamalli, joka keskittyy sovelluksen luotettavaan toimittamiseen eri testiympäristöihin ja lopulta julkaisuun tuotantoympäristössä. Yleensä ohjelmistoprojekteissa tämän vaiheen käytännön työn tekevät testaus- ja operointitiimit. Toistuva testaus sekä toimittaminen lisäävät projektitiimin luottamusta julkaisun tekemiseen virheettömästi. (Humble & Farley, 2010.) Jatkuvan toimittamisen ohella käytetään termiä jatkuva käyttöönotto (engl. Continuous Deployment). Se eroaa jatkuvasta toimittamisesta ainoastaan tavassa, jolla sovelluksen julkaisu tuotantoon käynnistetään. Jatkuvassa toimittamisessa julkaisun käynnistäminen tapahtuu aina manuaalisesti, kun taas jatkuvassa käyttöönotossa testit läpäissyt sovellus

toimitetaan automaattisesti tuotantoon asti. Jatkuvassa toimittamisessa käyttöönotto ja siten muutoksen tekeminen tuotantoon on paremmin kontrolloitu, ja se soveltuu useimpiin sovelluksen toimittamisen käyttötapauksiin. Jatkuvan käyttöönoton voi kuitenkin nähdä toteuttavan jatkuvan toimittamisen toimintamallin täydellisesti. (Chen, 2017, s. 73; Humble & Farley, 2010, s. 266–267.)

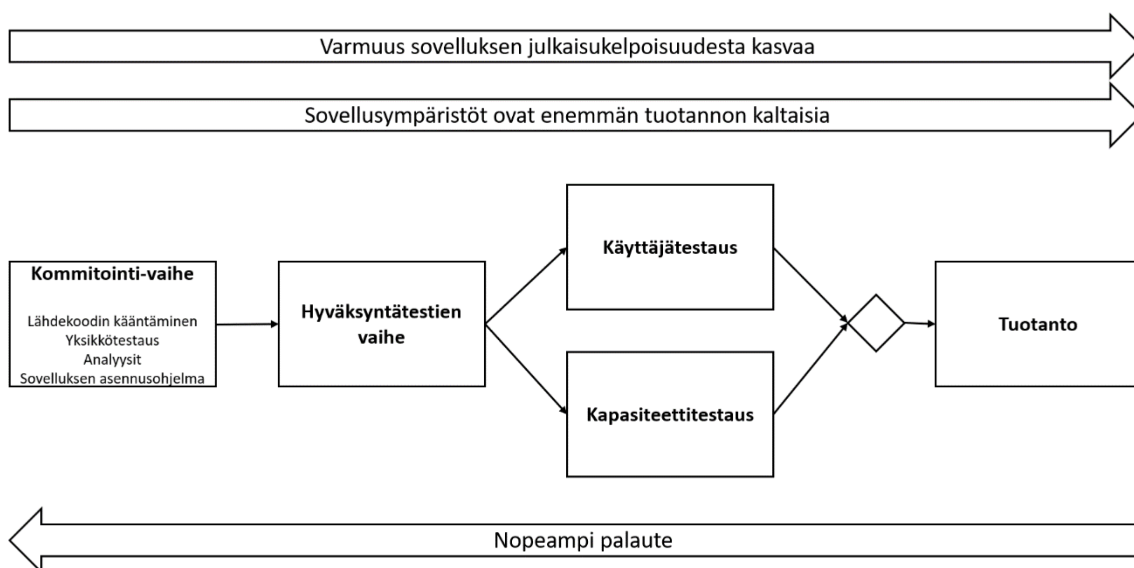
2.3.1 Nopea palaute ja lyhyempi kierrosaika

Jatkuva toimittaminen vähentää kierrosaikaa, joka kuluu asiakasvaatimuksesta päivityksen tekemiseen tuotannossa olevalle sovellukselle, josta asiakas voi jälleen tehdä uudet vaatimusmäärittelyt. Sen sijaan että valmistuneet toteutukset kasautuvat odottamaan sovelluksen versiopäivitystä tai julkaisuaikataulun mukaista päivityshetkeä, ne viedään tuotantoon heti niiden valmistuttua. (Chen, 2017, s. 73.) Jatkuvan toimittamisen ansiosta asiakaspalautteen saaminen käyttöönotetusta ympäristöstä nopeutuu siten huomattavasti (Chen, 2017, s. 73; Leppänen ja muut, 2015, s. 67; Olsson ja muut, 2012, s. 392–393). Tiheät julkaisut tuottavat lisäarvoa myös asiakkaan näkökulmasta, koska he näkevät konkreettisia edistysaskeleita nopeammin, mikä nostaa asiakastyytyväisyyttä (Leppänen ja muut, 2015, s. 67). Lisäksi Leppänen ja muut (2015, s. 67) havaitsivat, että nopeampi julkaisutahti paransi myös kehityksen ja operoinnin yhteistyötä, koska tiimit ovat jatkuvassa kontaktissa toisiinsa julkaisujen välillä. Aivan kuten jatkuvassa integroinnissa, nopea palaute on myös jatkuvan toimittamisen yksi keskeisistä hyödyistä.

2.3.2 Käyttöönottoputki

Käyttöönottoputkella (engl. Deployment Pipeline) tarkoitetaan työnkulkua, joka alkaa kehittäjän kommitoimasta koodipäivityksestä versionhallinnan päälinjaan. Alla olevassa kuvassa on esitetty malli käyttöönottoputkesta, sen sisältämistä vaiheista, sekä vaiheiden vaikutuksesta kehitystyöhön putken alkupäässä ja sovelluksen toimittamiseen putken loppupäässä. Päivitetty sovellus etenee nopeasti tehtävistä yksikkötesteistä

kattavampiin hyväksyntätesteihin ja lopulta kokonaisvaltaisiin käyttäjä- sekä kapasiteettitesteihin. Mikäli sovellus läpäisee kaikki testit, on se valmis julkaistavaksi tuotantoon. Mitä pidemmälle sovellus putkessa etenee, sitä enemmän eri testiympäristöt vastaavat tuotantoympäristöä. Samalla varmuus sovelluksen julkaisukelpoisuudesta tuotantoon kasvaa. Jokainen testivaihe antaa palautteen, mikäli sovelluksessa on virhe, eikä se siten läpäise testiä. Jotta käyttöönottoputki toimii tehokkaasti, tulee sovelluksen toimittaminen ja testaus automatisoida. Tosin toiminnallinen testaus voidaan tehdä myös manuaalisena käyttäjätestinä yhdessä asiakkaan kanssa. Automatisoidut tehtävät lisäävät varmuutta sovelluksen toimittamisen onnistumisesta, koska käyttöönotto niiden avulla on aina toistettavissa samalla tavalla. (Humble & Farley, 2010, s. 106–110.)



Kuva 2. Käyttöönottoputki sekä sen vaikutukset sovelluksen toimittamiseen ja kehittämiseen (Humble & Farley, 2010, s. 110).

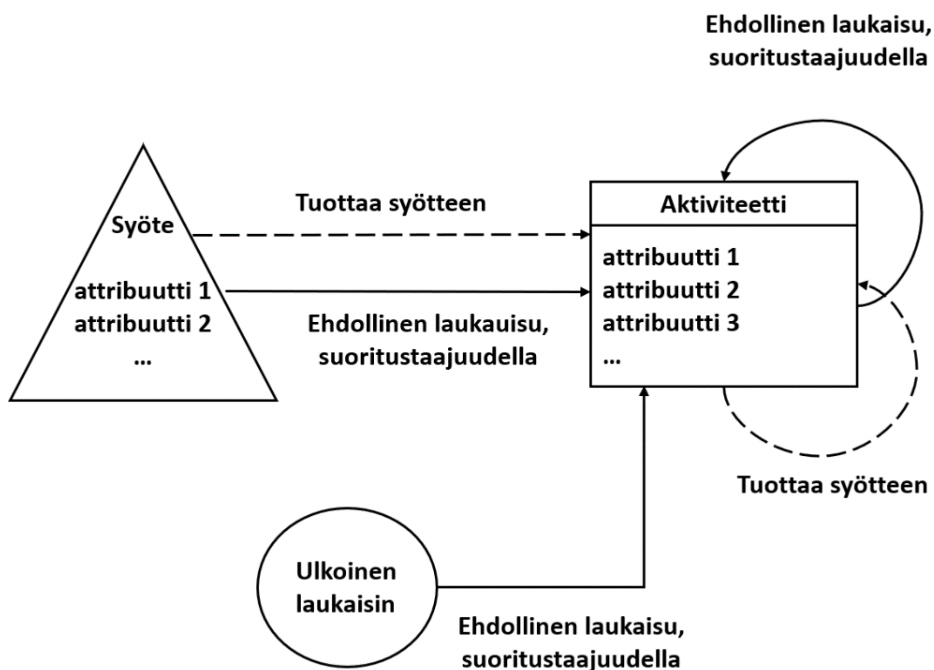
Sovelluksen toimituksen onnistuminen eri testiympäristöihin sekä tuotantoympäristöön on hyvä varmistaa automaattisilla käyttöönottotesteillä. Niillä todennetaan, että sovellus on käynnistynyt, se saa yhteyden tarvitsemiinsa ulkoisiin palveluihin (esim. tietokanta) ja että se on oikein konfiguroitu. (Humble & Farley, 2010, s. 89.) Mikäli julkaisu epäonnistuu, sovellus palautetaan edelliseen toimivaan versioon. Palautuksessa tulee huomioida, että mikäli julkaisu muuttaa sovelluksen käyttämää dataa, siitä pitää ottaa

varmuuskopio ennen julkaisua. Tällöin data säilyy palautuksessa muuttumattomana. Palautusta on myös syytä harjoitella, koska vaikka julkaisua harjoitellaan jatkuvan toimittamisen mallissa useita kertoja, palautuksia tehdään vain harvoin. (Humble & Farley, 2010, s. 259–260.) Julkaisun epäonnistumiseen vaikuttavan ongelman etsintä tulisi tehdä testiympäristössä. Näin tuotantoon ei jää konfiguraatioita, joilla ongelmaa on yritetty ratkaista. Lisäksi korjaus tulee varmasti dokumentoiduksi versionhallintaan. Kiireellisiäkään korjauksia ei tehdä suoraan tuotantoon, vaan ne menevät aina käyttöönottoputken testien läpi aivan kuten normaalit päivitykset (Humble & Farley, 2010, s. 265–266).

Käyttöönottoputkesta riippuen toimituksen ja testauksen käynnistäminen voi tapahtua joko automaattisesti tai manuaalisesti esim. yhdellä komentoriviltä ajettavalla komennolla. Kun käynnistykset tehdään manuaalisesti, muodostuu käyttöönottoputkesta imuohjautunut järjestelmä (engl. pull system), jossa testaus- ja operointitiimien jäsenet voivat ottaa käsiteltäväksi sovelluksen versioita sitä mukaa kun niitä on putkessa saatavilla (Humble & Farley, 2010, s. 106). Käyttöönottokomento tai -skripti kannattaa suunnitella niin, että samalla komennolla tehdään käyttöönotto sekä testaus- että tuotantoympäristöön. Eri ympäristöissä käytettävät asetukset komento käy lukemassa konfiguraatiotiedostoista. Komennot, skriptit ja konfiguraatiotiedostot säilytetään sekä ylläpidetään versionhallinnassa, jolloin ne testataan ohjelmakoodin ohella useaan kertaan. (Humble & Farley, 2010, s. 115–117.)

2.3.3 Käyttöönottoputken mallinnus

Ståhl ja Bosch (2014a) ovat luoneet meta-mallin, jonka avulla on mahdollista kuvata erilaisia jatkuvan integroinnin toteutuksia. Alkuperäistä mallia he vielä täydensivät tutkiessaan yrityksissä käytössä olevia työnkulkuja (Ståhl & Bosch, 2014b). Päivitetty malli on esitetty alla olevassa kuvassa, ja sitä on käytetty tässä tutkimuksessa jatkuvan integroinnin ja jatkuvan toimittamisen käyttöönottoputken mallintamiseen.



Kuva 3. Jatkuvan integroinnin työnkulun meta-malli (Ståhl & Bosch, 2014b, s. 57).

Meta-mallissa on kahdenlaisia komponentteja: solmuja, joiden välille muodostuu integraatiovuo, sekä solmujen sisältämiä attribuutteja. Solmut kuvaavat erilaisia tehtäviä integraatiossa. Niitä yhdistävät toisiinsa suunnatut kaaret, jotka käynnistävät seuraavassa solmussa kuvatut tehtävät. Mallia voi siten ajatella suunnattuna syklittömänä graafina. Solmut voidaan suorittaa joko peräkkäin, rinnakkain, tai ne voidaan ajastaa käynnistymään eri aikoihin. (Ståhl & Bosch, 2014a.)

Solmuja on kolmenlaisia: aktiviteettisolmuja, syötesolmuja sekä ulkoisia laukaisimia (engl. trigger). Aktiviteettisolmussa suoritetaan integraation tehtäviä, esim. ajetaan erilaisia testejä. Syötesolmu toimii lähteenä aktiviteettisolmun käyttämälle tiedolle, jota voi olla esim. lähdekoodi. Syöte voi myös käynnistää aktiviteettisolmun tehtävän suorittamisen. Ulkoinen laukaisin nimensä mukaisesti toimii aktiviteettisolmun käynnistäjänä. Kun aktiviteettisolmu saa tehtävänsä suoritettua, se voi itsessään käynnistää seuraavan aktiviteetin, ja samalla se voi olla myös tiedon lähde. Kaikissa tapauksissa käynnistäminen tapahtuu ehdollisena (esim. edellinen aktiviteetti onnistui tai epäonnistui), ja sille

voidaan määrittää suoritustaajuus (onko laukaisin manuaalinen vai ajastettu, tai miten usein lähdekoodia integroidaan päälinjaan). (Ståhl & Bosch, 2014a; 2014b.)

Syöte- ja aktiviteettisolmut sisältävät lisäksi attribuutteja. Syöte-solmujen attribuutteja ovat integraation alustus sekä integraation kohde -attribuutit. Alustus sisältää vaatimuksen esim. koodin katselmoinnista tai kehittäjän omalla koneella ajettavista testeistä, ennen kuin integraatio tehdään päälinjalle. Kohteena voi olla muita versionhallinnan haaroja kuin päälinja. Vaikka tällainen strategia on versionhallinnan näkökulmasta mahdollinen, johtaa se usein lopulta yhteensovittamisongelmiin päälinjalle integroitaessa, ja siten yleensä suositaan ainoastaan päälinjan käyttöä. (Ståhl & Bosch, 2014a.)

Aktiviteettisolmuissa käytetään laajuutta, ominaisuuksia ja tulosten käsittelyä kuvaavia attribuutteja. Laajuuden yhteydessä voidaan esim. määrittää, millaista testausta aktiviteetissa tehdään: yksikkötestausta, koodin analyysitestausta, toiminnallista ja ei-toiminnallista testausta tai hyväksyntätestausta. Ominaisuuksia kuvaavia attribuutteja ovat esim. tehtävän kesto, sovelluksen rakentamistiheys (kuinka usein sovellus käännetään ajettavaksi ohjelmaksi) ja aktiviteetin laukaisun määrittävät muuttujat (esim. rikkinäisessä päälinjassa aktiviteetin voi käynnistää ainoastaan korjauksen sisältävä koodipäivitys). Aktiviteetin lopputuloksesta riippuvia attribuutteja ovat esim. tuloksen määritelmä (onnistunut tai epäonnistunut) ja tuloksen raportointi (kenelle, miten ja milloin). (Ståhl & Bosch, 2014a.)

2.3.4 Käyttöönottoputken toteuttaminen

Humble ja Farley (2010, s. 133–137) esittävät viiden kohdan mallin käyttöönottoputken toteuttamiselle ohjelmistoprojektissa. Malli etenee kohta kohdalta seuraavaan vaiheeseen. Käyttöönottoputkea ei ole kuitenkaan tarkoitus tehdä kerralla valmiiksi, vaan siinä määritellyt tehtävät ja niiden käyttämät skriptit tarkentuvat ja päivittyvät projektin edetessä. Lisäksi, jos jokin tehtävä on täysin manuaalinen, se kannattaa pitää mukana

käyttöönottoputkessa. Tällöin sen kestoa voidaan mitata, ja jos se osoittautuu pullonkaulaksi, se kannattaa automatisoida. Viisi kohtaa ovat:

1. Mallinna arvoketju ja luo ajettava malli.
2. Automatisoi rakentamis- ja käyttöönottoprosessi.
3. Automatisoi yksikkö- ja koodianalyysitestit.
4. Automatisoi hyväksyntätestit.
5. Automatisoi julkaisu.

Kohdassa 1 mallinnetaan vaiheet sekä niiden sisältämät tehtävät. Uudessa projektissa ne eivät vielä sisällä vielä mitään toimenpiteitä, vaan ainoastaan varaavat paikkaa käyttöönottoputkessa. Sisältö täydentyy projektin edetessä. Niille on kuitenkin hyvä määritellä minimaalinen tehtävä, jolla käyttöönottoputken toimivuuden voi todentaa. Ohjelmistokehityksessä käytetään usein ”hello world” -ohjelmaa. (Humble & Farley, 2010, s. 133–134.)

Kohdassa 2 rakentamisprosessilla viitataan ohjelmiston kääntämiseen, jossa ohjelmakoodista muodostuu ajettavia ohjelmatiedostoja. Jatkuvan integroinnin ja toimittamisen prosessissa niitä kutsutaan myös artefakteiksi. Kääntäminen tehdään vain kerran, kun ohjelmakoodi on kommitoitu versionhallintapalvelimelle. Onnistuneen kääntämisen jälkeen artefakti tallennetaan projektin käytössä olevalle levytilalle, josta kääntämistä seuraavat vaiheet ja tehtävät voivat käydä sen vuorollaan noutamassa. Käyttöönottoja varten tulee eri testiympäristöihin asentaa laitteet ja ohjelmistot siten, että tekeillä oleva sovellus voidaan niissä testata. Käyttöönotto-skripteillä määritellään, miten sovellus asennetaan ja konfiguroidaan näihin ympäristöihin. Lisäksi määritellään käyttöönoton testaus-skriptit, joilla todennetaan että sovellus on asentunut oikein. (Humble & Farley, 2010, s. 134–135.)

Kohdan 3 kommitointivaiheen koodianalyysitestit ovat nopeasti määriteltävissä, mikäli projektissa käytetylle ohjelmointikielelle on saatavilla valmis työkalu. Vaiheen testit tulee ajaa nopeasti läpi, ja projektin edetessä sekä testattavan koodimäärän kasvaessa

tuleekin seurata, että testit eivät kasva liian pitkäkestoisiksi. Mikäli näin käy, tulee ne ajaa esim. rinnakkain usealla testipalvelimella (Humble & Farley, 2010, s. 135–136).

Kohdassa 4 otetaan käyttöön hyväksyntätestit. Projektin alussa kannattaa aloittaa etenkin ei-toiminnallisilla testeillä kuten kapasiteettitesteillä. Tällöin projekti pystyy alusta asti seuraamaan, miten hyvin ohjelma täyttää suorituskykyvaatimukset. Projektin edetessä toiminnalliset testit voi erottaa omaan vaiheeseen, jolloin virheiden jäljittäminen on helpompaa. (Humble & Farley, 2010, s. 136.)

Kohdan 5 määritellään tuotannossa käytettävät konfiguraatiotiedot versionhallintaan, ja hyödynnetään samaa käyttöönotto- ja testausskriptiä, joka oli määritelty kohdassa 2 (Humble & Farley, 2010, s. 135).

Chen (2017, s. 77–79) on havainnut, että jatkuvan toimittamisen toimintamallin hyväksyntä tapahtuu projektissa helpommin, kun käyttöönottoputkea voidaan päivittää ja täydentää projektin kuluessa. Aluksi projektille kannattaa tehdä käyttöönottoputkesta visuaalinen runko, jossa eri vaiheet on mallinnettu omille paikoilleen, vaikka niille ei vielä olisikaan toteutusta, kuten Humble ja Farley kohdassa 1 yllä ehdottavat. Projektin edetessä tiimi täydentää itsenäisesti puuttuvat kohdat. Käyttöönottoputken toteutukseen voi siten myös hyödyntää jatkuvan toimittamisen toimintamallia. Joka tapauksessa toimintamallin menestyksekkään käytön kannalta on oleellista, että kehitystiimi tietää, miten käyttöönottoputki toimii (Leppänen ja muut, 2015, s. 69–70).

2.4 Konfiguraation hallinta

Sovelluksen konfiguraatiotiedot ovat ajettavien ohjelmatiedostojen sekä sovelluksen sisältämän käyttäjädatan lisäksi tärkeä osa ohjelmistoa. Siksi niitä tulisi käsitellä kuten ohjelmistokoodia. Ne tulisi säilyttää versionhallinnassa, ja testata sovelluksen käyttöönottotestien yhteydessä. (Humble & Farley, 2010, s. 39.)

Koska testaus- ja tuotantoympäristöt poikkeavat yleensä jonkin verran toisistaan, ovat konfiguraatiotiedot ainakin osaksi ympäristökohtaista. Sovelluksen toimittamiseen tulee kuitenkin käyttää samoja skriptejä kohdeympäristöstä riippumatta, jolloin varmuus niiden toimivuudesta tuotantoon julkaisussa vahvistuu. Skriptit suunnitellaan siten, että ne voivat hakea oikeat asetustiedot ympäristön mukaan. Tiedot voivat olla samassa versionhallinnan säiliössä kuin sovellus, mutta koska niitä päivitetään eri tahtiin, suositus on tehdä niille oma säiliö. (Humble & Farley 2010, s. 41–43.) Konfiguraatiotietojen dokumentoinnissa tulee kuitenkin muistaa tietoturvanäkökulma, ja arkaluontoiset tiedot, kuten salasanat, on syytä säilyttää niille tarkoitetuissa järjestelmissä erillään projektin versionhallinnasta (Humble ja Farley 2010, s. 48; Johann, 2017). Jatkuvan integroinnin palvelin onkin usein tietoverkkoon tunkeutujien ensimmäisiä kohteita, joista etsiä salasanoja muihin järjestelmiin (Johann, 2017). Konfiguraatioiden testaus voidaan tehdä kahdessa vaiheessa. Ensimmäisessä vaiheessa testataan, että järjestelmät, joita sovellus konfiguroidaan käyttämään, ovat tavoitettavissa. Tämä voidaan tehdä testaamalla yhteys esim. ping-komennolla. Toisessa vaiheessa testataan, että sovelluksen toiminnallisuudet, joihin konfiguraatiot vaikuttavat, toimivat kuten niiden pitäisi. (Humble ja Farley 2010, s. 46.)

Konfiguraation hallinta ei rajoitu vain sovelluksen konfiguraatiotietoihin. Ympäristö, jossa sovellusta ajetaan, koostuu erilaisista komponenteista, jolloin myös niiden asetukset tulisi sisällyttää hallinnan piiriin. Näitä ovat esim. palvelinkäyttöjärjestelmät, tietokantapalvelimet ja tietoliikenneverkon laitteet. (Humble & Farley 2010, s. 50.) Koko sovellusympäristön hallinnalle konfiguraatiotietojen avulla käytetään termiä koodattu infrastruktuuri (engl. Infrastrucure as Code) (Fowler, 2016). Termillä korostetaan sitä, että sovellusympäristöä voidaan kuvata lähdekoodin tavoin, ja siten sitä voidaan käsitellä jatkuvan integroinnin ja jatkuvan toimittamisen menetelmillä (Fowler, 2016; Johann, 2017). Lisäksi koko ympäristö on toimitettavissa automaattisesti skriptien avulla, jolloin ongelmatilanteissa ympäristön provisiointi uudelleen konfiguraationhallinnasta voi osoittautua resurssien kannalta edullisemmaksi, kuin vanhan ympäristön korjaaminen. (Humble & Farley 2010, s. 49–54.) Myös turvallisuus paranee, koska kaikki voivat nähdä koodin, ja

siihen tehdyt muutokset ovat tarkistettavissa versionhallinnasta. Lisäksi testit voidaan kirjoittaa niin, että ne tarkistavat että järjestelmiin on konfiguroitu ainoastaan sallitut käyttäjätunnukset. (Johann, 2017.)

Muutokset testi- ja tuotantoympäristöjen konfiguraatioihin tehdään samalla tavalla. Olennaista on, että niitä ei tehdä ympäristöihin suoraan ohittamalla versionhallinta ja käyttöönottoputki. Riskinä on, että oikaisemalla tehdyt muutokset voivat aiheuttaa helpommin palvelukatkoja, sekä jättää ympäristöön konfiguraatioita, joista voi aiheutua ongelmia myöhemmin. (Humble ja Farley, 2010, s. 285–287). Versionhallinta mahdollistaa muutosten selvittämisen konfiguraatiotietojen osalta, ja käyttöönottoputken skripteistä on helppo jäljittää mitä toimenpiteitä ympäristöön on tehty (Johann, 2017). Fowler (2012) käyttää termiä lumihiutale palvelimille, joihin on tehty konfiguraatioita ohi versionhallinnan ja käyttöönottoputken. Tällaisen palvelimen uudelleen provisioiminen versionhallinnasta on hyvin vaikeaa. Lisäksi muutosten tekemisessä voi ilmetä yllättäviä ongelmia, kun ympäristön ajossa oleva konfiguraatio ei vastaakaan versionhallinnassa olevaa konfiguraatiota.

Sovellusympäristön päivittämien ja hallinta tapahtuu yhteistyössä kehitys- ja operointitiimien välillä. Yhteistyötä tarvitaan, koska tiimien tavoitteet ovat keskenään tarkasteltuna ristiriitaiset. Kehitystiimin tulee päivittää nykyistä ja toimittaa uutta ohjelmistoa mahdollisimman nopeasti, kun taas operointitiimi keskittyy ympäristön ylläpitoon ja sen vakaaseen toimintaan (Humble & Farley, 2010, s. 277–280). Jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin mukaisesti usein tehdyt pienet päivitykset pienentävät julkaisuun liittyviä riskejä. Siten kummankin tiimin onnistumismahdollisuudet tavoitteiden saavuttamisessa paranevat.

Ympäristön hallinnan näkökulmasta tulee huomioida tiettyjä operointitiimin tarpeita. Ne on hyvä käydä läpi yhdessä kehitystiimin kanssa jo projektin alussa, ja kirjata julkaisusuunnitelmaan, jota käytetään mm. käyttöönottoputken toteuttamisessa. Julkaisusuunnitelmaan kirjataan mm. mitä työkaluja käytetään skriptien luomiseen. Työkalujen

valinnassa on myös syytä huomioida, ovatko ne molemmille osapuolille ennestään tut-tuja, vai tarvitaanko niiden osalta koulutusta. Myös ympäristön monitorointijärjestelmän erityispiirteet, esim. mitä protokollia kyseinen järjestelmä käyttää valvottavien laitteiden ja sovellusten tilan tarkasteluun, kirjataan ylös. Julkaisusuunnitelman avulla kehitystiimi voi ottaa tuotannonaikaiset ylläpidon tarpeet huomioon heti projektin alusta alkaen. (Humble ja Farley, 2010, s. 281–283.)

3 Tutkimusmenetelmä

Tutkimus tehtiin käyttämällä suunnittelutieteellistä tutkimusmenetelmää. Ensimmäisessä kappaleessa käyn lyhyesti läpi, mitä on suunnittelutieteellinen tutkimus. Hevner ja muut (2004; 2007) ovat määritelleet tutkimuksen tekemiselle viitekehyksen sekä suositukset, jotka esittelen kappaleissa 3.2 ja 3.3. Viimeisessä kappaleessa käsitellään suunnittelutieteellisen tutkimusmenetelmän prosessi, jonka ovat toteuttaneet Peffers ja muut (2008) ja jota luvussa 4 kuvattu tutkimusprosessi noudattaa.

3.1 Suunnittelutieteellinen tutkimus

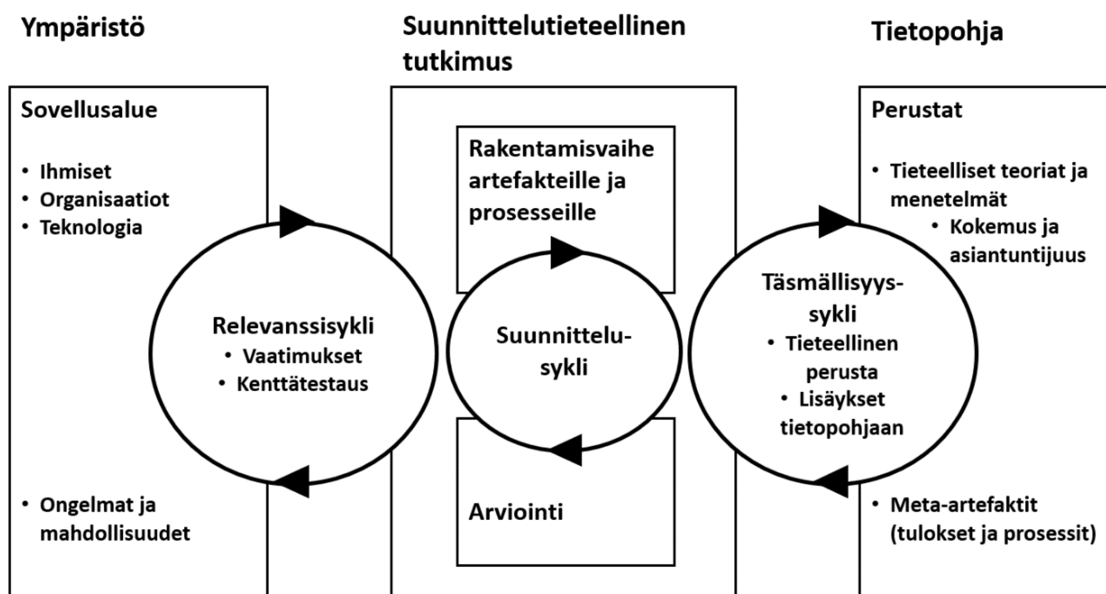
Suunnittelutieteellisessä tutkimuksessa pyritään tieteellisin menetelmin ratkaisemaan teknologiaan liittyviä ongelmia. Tutkimuksen tuloksena syntyy artefakti, jonka tarkoituksena on toteuttaa ongelman asettamat vaatimukset. Artefakteja on neljää tyyppiä: rakenteita (constructs), malleja (models), metodeja (methods) ja ilmentymiä (instantiations). Rakenteilla tarkoitetaan sanastoa, joiden avulla käsitteellistetään eri tieteenaloille oleellista tieteellistä termistöä. Tietokannoissa käytetty relaatiomalli on esimerkki rakenteesta. Mallit ovat kokoelma ehdotuksia tai väitteitä, joilla kuvataan rakenteiden välisiä suhteita. Esimerkkinä tietokantojen mallinnuksessa relaatioiden välisiä suhteita kuvataan ER-kaaviolla (engl. Entity Relationship). Metodit ovat rakenteisiin ja malleihin perustuva kokoelma vaiheita, esim. algoritmi tai ohjeistus, jolla suoritetaan tietty tehtävä. Ilmentymä on artefaktin toteutus sille tarkoitetussa ympäristössä. Se muodostuu rakenteista, malleista ja metodeista, joiden avulla on toteutettu toimiva kokonaisuus. Sen avulla osoitetaan rakenteiden, mallien ja metodien toimivuus tutkimusongelman ratkaisuna. (March & Smith, 1995.)

March ja Smith (1995, s. 258–259) esittävät suunnittelutieteelliselle tutkimukselle kaksi vaihetta: rakenna ja arvioi. Rakentamisvaiheessa suunnitellaan ja rakennetaan artefakti. Arviointivaiheessa kehitetään tutkimusongelman tavoitteiden mukainen mittaristo ja mittausprosessi, joiden avulla artefaktin toimivuus ongelman ratkaisuna voidaan

arvioida. Vaihe on oleellinen, sen puuttuessa on mahdotonta tieteellisesti arvioida rakentamisvaiheessa tehtyä tutkimustyötä.

3.2 Suunnittelutieteellisen tutkimuksen viitekehys

Hevner ja muut (2004, s. 79) esittävät suunnittelutieteelliselle tutkimukselle käsitteellisen viitekehyksen, joka koostuu kolmesta tutkimusongelman kanssa vuorovaikutuksessa olevasta elementistä: ympäristö, tietopohja ja suunnittelutieteellinen tutkimus. Vuorovaikutus elementtien ja tutkimuksen kahden aktiviteetin, rakentaminen ja arviointi, välillä tapahtuu sykleissä. Viitekehys on esitetty alla olevassa kuvassa, ja sen sisällöstä on kerrottu tarkemmin kappaleissa 4.2.1 ja 4.2.2.



Kuva 4. Suunnittelutieteellisen tutkimuksen viitekehys (Hevner, 2007, s. 88).

3.2.1 Viitekehyksen elementit

Ympäristö koostuu ihmisistä, organisaatioista ja teknologioista. Lisäksi se sisältää ongelmia ja mahdollisuuksia, jotka on määritelty ja arvioitu organisaation strategian sekä

prosessien kautta, ja jotka ovat muovautuneet kunkin ympäristön ihmisten ominaisuuksien, kyvykkyyksien ja roolien vaikutuksella. Lisäksi ne asemoidaan organisaation nykyisen käytössä olevan teknologian ja sen kehittämismahdollisuuksien mukaan. Tällä tavoin ongelmasta tai mahdollisuudesta muodostuu tutkijan näkökulmasta merkityksellinen suunnittelutieteellinen tutkimusongelma. (Hevner ja muut, 2004, s 79.)

Ongelman tai mahdollisuuden ratkaisuun suunnittelutieteellinen tutkimus käyttää tietopohjaa, joka koostuu tieteellisistä perusteista sekä menetelmistä, ajantasaisimmasta asiantuntijuudesta ja kokemuksesta tutkimusongelman alueella, sekä olemassa olevista meta-artefakteista eli rakenteista, malleista, metodeista ja ilmentymistä. Perusteita, esim. teorioita ja viitekehyksiä, käytetään tutkimuksen suunnitteluvaiheessa artefaktin suunnitteluun. Menetelmiä, esim. datan analysointitekniikoita, mittaustapoja ja hyväksyntäkriteerejä, käytetään arviointivaiheessa tapahtuvan artefaktin mittausprosessin suunnitteluun. (Hevner ja muut, 2004, s. 80, Hevner, 2007, s. 89)

Varsinainen suunnittelutieteellinen tutkimus tapahtuu kahdessa aktiviteetissa, rakentamis- ja arviointiaktiviteetissa. Ne ovat vastaavat kuin kappaleessa 4.1 kuvatut March ja Smithin (1995) määrittämät tutkimuksen vaiheet.

3.2.2 Viitekehyksen syklit

Suunnittelutieteellisen tutkimuksen aktiviteettien, sekä tutkimuksen, tietopohjan ja ympäristön välinen vuorovaikutus tapahtuvat Hevnerin (2007) mukaan kolmessa syklissä: relevanssisykli, täsmällisyyssykli ja suunnittelusykli.

Relevanssisykli käynnistää suunnittelutieteellisen tutkimuksen. Siinä määritellään vaatimukset tutkimukselle, mitä ympäristössä muodostunutta ongelmaa tai mahdollisuutta lähdetään tutkimaan. Lisäksi tutkimusongelman perusteella määritellään hyväksyntäkriteerit, joilla tuloksena syntyvä artefakti arvioidaan. Nämä ovat relevanssisyklin syötteet ympäristöstä tutkimukselle. Tutkimuksen tulokset testataan annetussa ympäristössä.

Testausmenetelmä määritellään tutkimuksen rakentamisvaiheessa, ja yhdessä artefaktin kanssa ne toimivat syötteenä tutkimukselta ympäristölle. Testauksen tulokset osoittavat, onko tutkimuksessa syntynyt artefakti toimiva ympäristössään. Se voi olla ominaisuuksiltaan tai laadultaan puutteellinen. Myös tutkimukseen syötetyt vaatimukset voivat olla virheellisiä tai puutteellisia, jolloin artefakti täyttää vaatimukset mutta ei ratkaise alkuperäistä ongelmaa. Kummassakin tapauksessa vaatimuksia tulisi tarkentaa, ja käynnistää relevanssisyklillä tutkimus uudelleen. (Hevner, 2007, s. 88–89.)

Täsmällisyyssyklissä suunnittelutieteellinen tutkimus hyödyntää tutkimusongelman aiheeseen liittyvää tietopohjaa, joka toimii siten syklin syötteenä tutkimuksen suunnittelulle (Hevner, 2007, s. 89–90). On oleellista, että tutkimus perustuu, ja siinä viitataan tieteellisiin lähteisiin. Näin varmistetaan, että tutkimuksessa syntyy uusi innovaatio, ja että tutkimuksen tulokset osaltaan täydentävät olemassa olevaa tietopohjaa. Mikäli tietopohjan parhaita käytäntöjä hyödynnetään sellaisenaan ympäristön asettaman ongelma ratkaisuun, on kyseessä rutiini suunnittelu, mikä ei sisällä tieteellistä panosta (Hevner ja muut, 2004, s. 81). Siten täsmällinen tutkimus edellyttää, että tutkija kykenee hyödyntämään sopivia teorioita ja metodeja tutkimuksen suunnittelu- ja arviointivaiheissa. Hevner (2007, s. 90) kuitenkin toteaa, että tiedeyhteisössä esiintyvä vaatimus tutkimuksen perustamisesta pelkästään tutkimuspohjan teoriasta löytyviin ideoihin ei ole välttämätöntä, vaan voi olla jopa vahingollista tieteellisen kehityksen kannalta. Sen sijaan tutkimuksen perustana voi hyödyntää uusia ajatuksia useista eri lähteistä, esim. relevanssisyklissä tunnistetuista monipuolisista ongelmista ja mahdollisuuksista sekä olemassa olevista artefakteista. Täsmällisyyssyklin syötteet tietopohjaan ovat tutkimuksessa syntyneet lisäykset olemassa oleviin teorioihin ja metodeihin, uudet artefaktit tai opitut kokemukset tutkimuksen suorittamisesta ja testaamisesta sen kohdeympäristössä. (Hevner, 2007, s. 90.)

Suunnittelusyklissä tapahtuu varsinainen tutkimustyö. Relevanssisyklissä ympäristöstä määritellylle ongelmalle valmistuu rakentamisvaiheessa ratkaisuksi artefakti. Samassa vaiheessa tapahtuu myös tutkimuksen arviointivaiheen mittariston ja mittausprosessin

suunnittelu. Apuna käytetään tietopohjaa täsmällisyyssyklistä. Arviointivaiheessa artefakti testataan huolellisesti, ja tarvittaessa sen kehittämistä jatketaan suunnitteluvaiheessa. Tutkimustyö etenee siten suunnittelusyklissä näiden kahden vaiheen välillä vuorotellen. Vasta täsmällisesti suunniteltu ja perusteellisesti testattu artefakti voidaan palauttaa relevanssisyklissä sille tarkoitettuun ympäristöön testattavaksi, sekä syöttää täsmällisyyssyklissä uutena tietona tietopohjaan. (Hevner, 2007, s. 90–91.)

3.3 Suositukset suunnittelutieteellisen tutkimustyön tekemiselle

Hevner ja muut (2004) ovat muodostaneet seitsemän suositusta, jotka auttavat tutkijaa huomioimaan suunnittelutieteellisen tutkimuksen vaatimuksia. Suositukset on esitetty alla olevassa taulukossa. Tutkimuksen aihe vaikuttaa siihen, mikä on paras tapa niiden noudattamiseen, ja tutkijasta riippuu, miten suosituksia on hyödynnetty. Joka tapauksessa niiden käyttäminen ei ole välttämätöntä, mutta niiden avulla tutkimuksesta muodostuu kokonaisvaltainen. Tässä tutkimuksessa suosituksia on käytetty apuna tutkimusprosessissa.

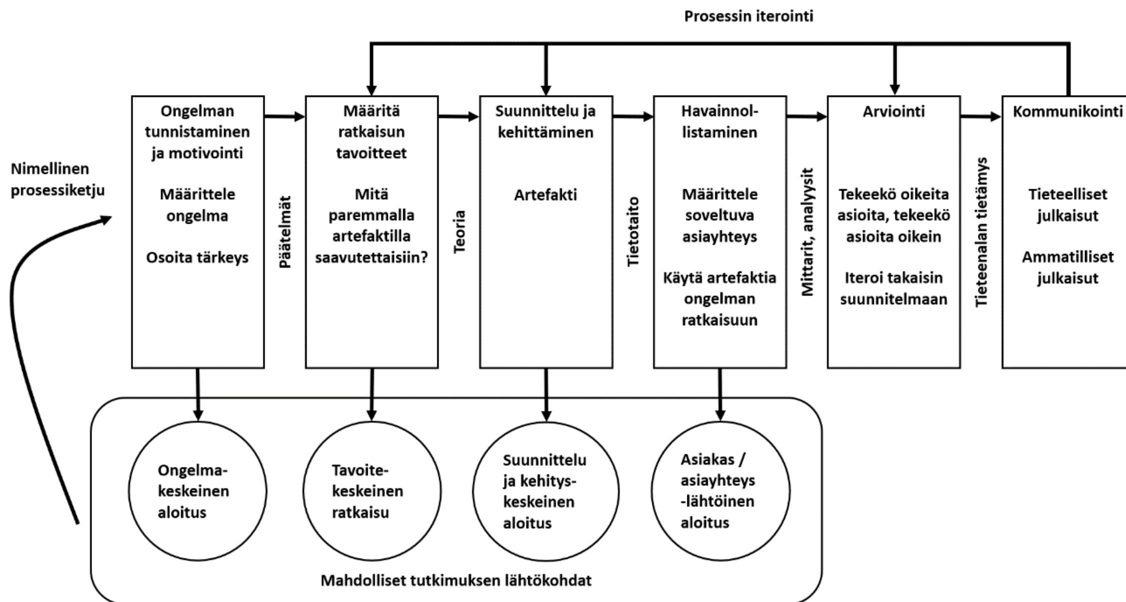
Taulukko 1. Suunnittelutieteellisen tutkimuksen suositukset (mukaillen Hevner ja muut, 2004, s. 83).

Suositus	**Kuvaus**
1) Tuloksena artefakti	Suunnittelutieteellisen tutkimuksen tuloksena tulee muodostua toteuttamiskelpoinen artefakti.
2) Ongelman olennaisuus	Suunnittelutieteellisen tutkimuksen tarkoituksena on kehittää teknologiaan perustuva ratkaisu tärkeään ja oleelliseen ongelmaan.
3) Suunnittelun arviointi	Artefaktin käytettävyys, laatu ja tehokkuus tulee täsmällisesti osoittaa huolellisesti suoritetuilla arviointimenetelmillä.

4) Tutkimuksen tieteellinen tuottavuus	Suunnittelutieteellisen tutkimuksen tulee luoda selkeitä ja toistettavia tuotoksia artefaktina, tietopohjan perusteina ja/tai tietopohjan menetelminä.
5) Tutkimuksen täsmällisyys	Suunnittelutieteellisen tutkimus nojaa täsmällisiin tutkimusmetodeihin sekä artefaktin rakentamis- että arviointivaiheessa.
6) Suunnittelu etsintäprosessina	Toimivan artefaktin suunnittelu vaatii käytettävissä olevien keinojen hyödyntämistä ongelmaympäristön sääntöjä noudattaen, jotta haluttu lopputulos voidaan saavuttaa.
7) Tutkimuksen kommunikointi	Suunnittelutieteellinen tutkimus tulee huolellisesti esittää sekä teknologiaan että johtamiseen suuntautuneelle yleisölle.

3.4 Suunnittelutieteellisen tutkimusmenetelmän prosessi

Peffers ja muut (2008) esittävät suunnittelutieteellisen tutkimuksen tekemiselle kokonaisvaltaisen menetelmän, jonka he ovat kuvanneet DSRM (Design Science Research Method) prosessimallina (ks. kuva 5). Se on tarkoitettu nimelliseksi malliksi, jota suunnittelutieteellisessä tutkimuksessa ei tarvitse välttämättä noudattaa, mutta aivan kuten kappaleen 4.3 suositukset, malli auttaa huomioimaan tutkimukselle olennaiset vaiheet.



Kuva 5. DSRM prosessimalli (Peffers ja muut, 2007, s. 88).

Vaihe 1: Ongelman tunnistaminen ja motivointi. Tutkittava ongelma tulee määritellä tarkasti. Se kannattaa pilkkoa mahdollisimman pieniksi käsitteiksi, jolloin monimutkaisenkin ongelman eri osat tulevat huomioiduksi kokonaisvaltaisesti (Peffers ja muut, 2008, s. 52–55.) Koska suunnittelutieteellisen tutkimuksen tulisi kohdistua tärkeiksi koettuihin ongelmiin, on ratkaisun tärkeys hyvä perustella. Se motivoi sekä tutkijaa että tutkimuksen yleisöä etsimään ratkaisua ja hyväksymään tutkimuksen tulokset, sekä auttaa ymmärtämään tutkijan ongelmasta tekemiä päätelmiä. (Hevner ja muut, 2004; Peffers ja muut, 2008.)

Vaihe 2: Määritä ratkaisulle tavoitteet. Ratkaisun tavoitteet tulee johtaa ongelman määrittelystä sekä rajata ne siten, että ne ovat mahdollisia ja toteuttamiskelpoisia. Tavoitteet voivat olla kvantitatiivisia (esim. millä arvoilla ratkaisu on parempi verrattuna olemassa oleviin ratkaisuihin) tai kvalitatiivisia (esim. kuvaus miten ratkaisu vastaa ongelmaan, jota ei aiemmin ole ratkaistu). Määrittelyssä tulee siten huomioida nykyisten, olemassa olevien ratkaisujen tehokkuus. (Peffers ja muut, 2008, s. 55.)

Vaihe 3: Suunnittelu ja kehittäminen. Tämän vaiheen tuloksena syntyy artefakti, josta ilmenee suunnitteluvaiheessa tehty tutkimustyö (Peffers ja muut, 2008, s. 55). Vaihe on March ja Smithin (1995) esittämä rakentamisvaihe, jossa hyödynnetään tietopohjan teoriaa artefaktin suunnitteluun siten, että vaiheessa 2 määritetyt tavoitteet saavutetaan. Myös mittaristo ja mittausprosessi suunnitellaan, jotta artefaktin toimivuus tavoitteiden suhteen voidaan todentaa.

Vaihe 4: Havainnollistaminen. Artefaktin käyttökelpoisuus yhden tai useamman ongelman osan ratkaisuna tulee havainnollistaa. Sen voi tehdä esim. kokeena, simulaationa, tapaustutkimuksena, näyttönä tai muuna vastaavana menetelmänä. Havainnollistamisessa on oleellista tietää, miten artefaktia pystyy käyttämään ongelman ratkaisussa. (Peffers ja muut, 2008, s. 55.)

Vaihe 5: Arviointi. Artefaktin soveltuvuus ongelman ratkaisuksi testataan sille tarkoitetussa ympäristössä (Peffers ja muut, 2008, s. 56). Testaus tehdään suunnittelusyklissä valmistuneen testaussuunnitelman mukaisesti (Hevner, 2007). Mikäli artefakti todetaan laadultaan liian heikoksi, tai se ei täytä sille tutkimuksen alussa asetettuja tavoitteita, tehdään päätös tavoitteiden päivittämisestä sekä suunnittelu- ja kehitysvaiheen aloittamisesta uudelleen, tai jätetään kehitys myöhemmille projekteille ja siirrytään kommunikaatiovaiheeseen (Hevner ja muut, 2004; Peffers ja muut, 2008, s. 56).

Vaihe 6: Kommunikointi. Tutkimus tulee esittää muille tutkijoille sekä sen aihealueen asiantuntijoille. Esitelmän sisältöön vaikuttavat kaikki DSRM-prosessin vaiheet, jolloin siinä käsitellään mm. ongelma ja sen tärkeys, artefaktin uutuus, käytettävyys ja hyödyllisyys, suunnittelun täsmällisyys. Oppilaitoksissa päättötyö on tällainen esitelmä. (Peffers ja muut, 2008, s. 56.)

4 Toimintamallin tutkimusprosessi

4.1 Ongelman tunnistaminen ja motivointi

Sovelluskehityksessä käytetyt toimintamallit ja työkalut ovat tulossa käyttöön myös tietoverkkojen hallintaan. Se miten konfiguraationhallintatyökaluilla pystytään automaattisesti hallinnoimaan ympäristöä, kuvastaa tämän päivän ajattelutapaa, jossa toistuvia tehtäviä pyritään automatisoimaan tietokoneiden käsiteltäväksi, ja siten vapauttamaan työaikaa ja tehostamaan työntekoa vaikeammin automatisoitavissa tehtävissä. Ylipäätään suuntauksena on, että automatiikka ja robotiikka ovat tulossa käyttöön useilla eri liiketoimintojen alueilla, ja toimintavarmat tietoverkot mahdollistavat osaltaan tätä kehitystä. Myös työn tekeminen on tulevaisuudessa yhä vähemmän paikkaan sidottua, työntekijä voi suorittaa työtehtävänsä joko työpaikalla, kotonaan tai vaikka kahvilassa. Hän kuitenkin tarvitsee pääsyn yrityksen tietojärjestelmiin, olivat ne sitten pilvessä tai yrityksen sisäverkossa. Tietoverkkojen tulee olla joustavia, mutta myös tietoturvallisesti konfiguroituja. Jotta nämä vaatimukset toteutuvat, tulee verkkojen provisiointi kyetä tekemään täsmällisesti, ja siinä verkonhallinnan tehtävien automatisaatiolla on keskeinen tehtävä. (Cisco Systems, 2019b.)

Konfiguraationhallintatyökalut käyttävät skriptejä, lyhyitä ohjelmakoodeja konfiguraatioiden automaattiseen provisiointiin verkkolaitteisiin. Skriptejä ja konfiguraatioita tulisi säilyttää ja hallita jossain muualla kuin verkkolevyn kansioissa. Versionhallintatyökalut ovatkin niille parempi säilytyspaikka. Lisäksi ne mahdollistavat jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallien mahdollistavien työkalujen käytön. (Juniper Networks, 2019.)

Alfa-yrityksen palvelutuotanto-organisaatiossa on myös tunnistettu yllä mainittu muutos, ja sen mukanaan tuoma ongelma. Koska mallit ja työkalut ovat samoja kuin sovelluskehityksessä on käytetty, on ohjelmoinnin periaatteiden ymmärtäminen ja ohjelmointiosaaminen oleellinen osa niiden hyödyntämistä. Alfa-yrityksessä ei kuitenkaan ole kokemusta sovelluskehityksestä. Siksi yrityksessä haastateltiin automaatiota omassa

sovellusympäristössään hyödyntävää ohjelmistoyritystä toimintatavoista ja työkaluista. Tämän perusteella yhdeksi tärkeäksi kokonaisuudeksi nousi jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin hyödyntäminen osana muutosta. Sen todettiin painottavan sovelluskehityksen toimintatapoja, kuten versionhallintaa, testaamista ja tiimin yhteistyötä, osana mallin työnkulkua. Siten toimintamallin avulla kaikki muu konfiguraationhallinnan automatisointiin liittyvä testaus pysyy keskitetysti yhden työkalun hallinnassa. Toimintamalli on kuitenkin laaja, ja se itsessäänkin koostuu kahdesta eri mallista. Siksi alfa-yrityksessä todettiin, että tuleekin selvittää, mistä toimintamallissa on kysymys, mitä se pitää sisällään ja miten se vaikuttaa nykyiseen toimintatapaan.

Tässä tutkimuksessa selvitetään jatkuvan integroinnin ja jatkuvan toimittamisen käyttöönottoon verkonhallinnassa liittyviä aiheita kysymällä: Miten hyödyntää jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia verkkolaitteiden hallintaan lähiverkon palvelutuotannossa? Verkonhallinnan näkökulmasta mm. erilaiset verkkoympäristöjen valvontatyökalut ovat osa hallintakokonaisuutta, ja myös niissä on omat mahdollisuutensa tehdä ohjelmallisia konfiguraatioita automaattisesti. Alfa-yrityksen palvelutuotannossa tiimit ovat organisoituneet siten, että osa tiimeistä työskentelee jatkuvan palvelun ylläpidossa, osa rakentaa asiakkaille uusia palveluita ja osa rakentaa uudet palvelut toimittajan ympäristöön. Tutkimuksessa toimintamallin hyödyntäminen verkonhallinnan työkalujen osalta on rajattu verkkolaitteiden konfiguraatioiden tekemiseen.

4.2 Artefaktin tavoitteet

Tavoitteena on kuvata jatkuvan integroinnin ja jatkuvan toimittamisen työnkulku sovitettuna verkkolaitteiden konfiguraationhallintaan. Työnkulun tulee esittää toimintamallista saavutettavat hyödyt, mutta myös haasteet tai havaitut kohdat, joita sen käyttöönotossa on syytä ottaa huomioon. Artefaktin perusteella organisaation tulee pystyä tekemään päätös, miten verkonhallintatyön muutoksen kanssa kannattaa edetä, sekä miten jatkuva integroiminen ja jatkuva toimittaminen ovat osana tätä muutosta. Tarkoitus on, että artefaktia voi hyödyntää joko sellaisenaan tai soveltaen verkkolaitteiden hallintaan.

4.3 Suunnittelu ja kehitys

Artefaktin kehityksessä käytettiin Humblen ja Farleyn (2010, s. 133–137) viiden kohdan mallia käyttöönottoputken toteuttamiselle. Mallin kohdat ovat:

1. Mallinna arvoketju ja luo ajettava malli.
2. Automatisoi rakentamis- ja käyttöönottoprosessi.
3. Automatisoi yksikkö- ja koodianalyysitestit.
4. Automatisoi hyväksyntätestit.
5. Automatisoi julkaisu.

Koska mallia on tarkoitus toteuttaa sykleissä, se sopi luontevasti käytettäväksi Hevnerin (2007, s. 88) esittämässä suunnittelutieteellisen tutkimusmenetelmän suunnittelusyklissä. Rakentamisvaiheessa teoriapohjaa hyödynnettiin käyttöönottoputken toteutukseen, ja sen toimivuus testattiin arviointivaiheessa. Seuraavaksi käyn läpi miten mallia hyödynnettiin ensimmäisen suunnittelusyklin aikana.

Kohdassa 1 aloitettiin käyttöönottoputken mallintaminen. Jatkuvan integroinnin ja jatkuvan toimittamisen teoriasta koottiin mallintamisen tueksi alla olevassa taulukossa 1 olevat, aihealueittain ryhmitellyt tavoitteet, jotka valmiin käyttöönottoputken tulisi toteuttaa. Niitä käytettiin seuraavissa sykleissä tehdyssä suunnittelussa, jossa verkkolaitteiden konfiguraationhallintaa sovitettiin automaattisesti suoritettavaan integroinnin ja toimittamisen malliin. Hevnerin (2007, s. 88) suunnittelusyklien arviointivaiheessa havainnoitiin, vastasiko sen hetkinen artefakti syklille asetettuja tavoitteita.

Kohdassa 2 rakennettiin ympäristö, jossa tutkimus voitiin suorittaa. Se toteutettiin tutkijan omalle tietokoneelle virtualisointitekniikoita hyödyntäen. Merkittävimmät komponentit ympäristössä olivat jatkuvan integroinnin palvelin, konfiguraationhallintatyökalu sekä tietoliikenneverkko. Esittelen tutkimuksessa käytetyt työkalut sekä verkkotopologian kappaleissa 4.3.1 – 4.3.4.

Kohdassa 3 selvitettiin, onko edellisessä kohdassa valitulle konfiguraationhallintatyökalulle olemassa valmiiksi tehtyjä sovelluksia, joita voisi hyödyntää yksikkö- ja koodianalyysitestauksessa. Työkalusta löytyi tähän sopiva valmis ominaisuus, ja lisäksi löytyi erillinen sovellus, joita voi käyttää samaan tapaan kuin tässä kohdassa määritellyt automatisoitavat testit. Esittelen löydökset konfiguraationhallintatyökalun yhteydessä kappaleessa 4.3.2.

Kohdan 4 testaus ja kohdan 5 julkaisu olivat vielä tässä kohtaa vain paikanpitäjinä käyttöönottoputken ensimmäisessä versiossa.

Taulukko 2. Rakentamisvaiheessa käyttöönottoputkelle asetetut tavoitteet.

Aihe	**Tavoite**	**Lähde**
Nopea palaute	Palaute testauksesta CI:ssä ja CD:ssä.	Humble & Farley, 2010 Ståhl & Bosch, 2014a
	Jatkuvalla toimittamisella saavutettava nopea kierrosaika ja asiakaspalaute.	Chen, 2017 Leppänen ja muut, 2015 Ohlsson ja muut, 2012
	Uusilla palveluilla oltava lyhyet kehityssyklit.	Fowler, 2006
	Työnkulun on oltava välitön.	Ståhl & Bosch, 2014b
	Työnkulun on oltava läpinäkyvä.	Ståhl & Bosch, 2014b
	Tiimin yhteistyöllä rakentama käyttöönottoputki.	Chen, 2017 Humble & Farley, 2010 Leppänen ja muut, 2015
Versionhallinta	Käytä vain päälinjaa.	Fowler, 2006 Humble & Farley, 2010 Ståhl & Bosch, 2014a
	Hyödynnä kommunikointivälineenä.	Fowler, 2006 Humble & Farley, 2010
	Päälinjan on pysyttävä eheänä.	Fowler, 2006 Humble & Farley, 2010 Ståhl & Bosch, 2014a

Testaus	Suoritusaika voi olla muutamia minuutteja.	Fowler, 2006 Humble & Farley, 2010
	Yksikkötestit	Humble & Farley, 2010
	Komponenttitestit	Humble & Farley, 2010
	Hyväksyntätestit	Humble & Farley, 2010
Palautus	Epäonnistunut toimitus tulee pystyä palauttamaan.	Humble & Farley, 2010
	Palautuksen toimivuus testattava säännöllisin väliajoin.	Humble & Farley, 2010
Konfiguraationhallinta	Yleiskäyttöiset skriptit siten, että samoilla skripteillä toimitetaan sekä testi- että tuotantoympäristöön.	Humble & Farley, 2010
	Kaksivaiheisella testauksella varmistetaan laitteiden tavoitettavuus ja palveluiden oikeanlainen toiminta.	Humble & Farley, 2010

4.3.1 Jatkuvan integroinnin ja jatkuvan toimittamisen työkalut

Jatkuvan integroinnin ja jatkuvan toimittamisen työkaluna käytettiin GitLabia. Se on sovelluskehitykseen tarkoitettu projektinhallintatyökalu, jonka yksi oleellinen osa on CI/CD-palvelinominaisuudet. Jotkin GitLabin sisältämistä työkaluista ovat maksullisia, mutta tutkimuksen toteuttamiseen ilmaisen version ominaisuudet riittivät mainiosti. GitLabin CI/CD-palvelimen lisäksi tuli asentaa myös erillinen GitLab Runner -palvelin, jonka tehtävänä on suorittaa käyttöönottoputki. Toteutusohjeet se lukee käyttöönottoputken kuvaustiedostosta, joka tallennetaan versionhallintaan osana muita projektiin kuuluvia tiedostoja (mm. verkkolaitteiden konfiguraatiotiedot). GitLab:ssa käyttöönottoputken kuvaamiseen käytettävä tiedosto on nimeltään .gitlab-ci.yml, ja se sijoitetaan versionhallinnan säiliön juurihakemistoon. (GitLab, 2019.) Tiedostopäätteestä nähdään, että kuvauskielenä on YAML (YAML Ain’t Markup Language). Se on tietorakenteiden kuvaukseen suunniteltu kieli, jonka sisältö on visuaalisesti ihmisen helposti ymmärrettävissä. Sitä

hyödynnetään myös tiedon välittäjänä eri ohjelmointikielien välillä, mistä esimerkkinä juurikin konfiguraatiotietojen kuvaaminen. (Oren ja muut, 2009.)

4.3.2 Konfiguraationhallintatyökalu

Konfiguraationhallintatyökaluna tutkimuksessa käytettiin Ansiblea. Sitä käytetään palvelinympäristöjen konfiguraationhallintaan, ja nykyään siihen on saatavilla valmiita moduuleja myös verkkolaitteiden provisiointiin (Ansible, 2020a). Muun muassa tutkimuksessa käytetylle Aristan EOS-ohjelmistolle Ansiblessa oli valmiit moduulit.

Ansiblella verkkolaitteiden konfiguraatio tallennetaan määrittelytiedostoihin. Laitteiden provisiointi määrittelyiden perusteella tapahtuu pelikirjojen (engl. playbook) avulla. Niissä on kuvattu (YAML-kuvauskielellä), mitä tehtäviä pelikirja suorittaa, jotta Ansible saa konfiguroitua kohteen haluttuun tilaan. Tietolähteenään tehtävät käyttävät mm. konfiguraatioiden määrittelytiedostoja. Tehtävän kuvauksessa käytetyt moduulit, esim. tutkimuksessa käytetyt Aristan EOS-moduulit, suorittavat varsinaiset komennot kohdeympäristössä. (Ansible, 2020a.) Kohdelaitteiden konfiguraatioiden luomiseen määrittelytiedostoista käytettiin lisäksi Jinja2-mallintamista (engl. templating), jota Ansiblessa on mahdollista käyttää sellaisenaan pelikirjojen kuvauksessa Ansible, 2020b).

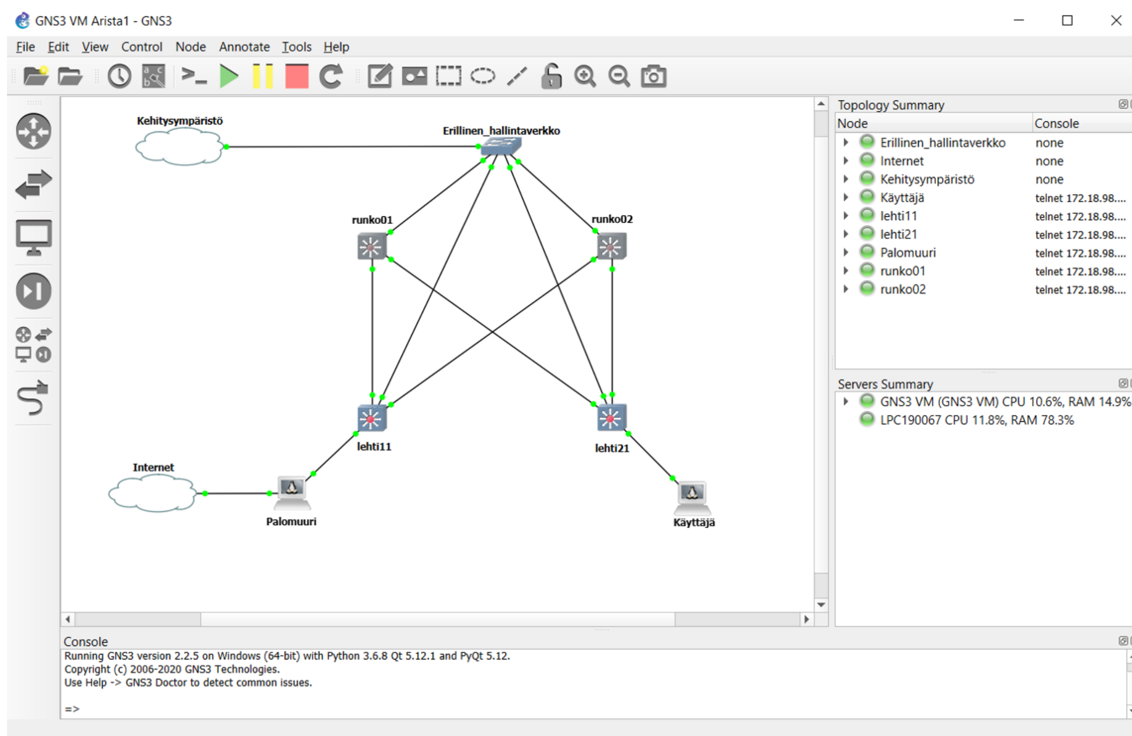
Käyttöönottoputken toteutusmallin kohdassa 3 selvitettiin, onko Ansiblessa valmiina työkaluja, joita voisi hyödyntää yksikkö- ja koodianalyysitestauksessa. Yksikkötestaukseen ei ollut työkalua, jota olisi voinut käyttää pelikirjan testaukseen. Sen pystyi kuitenkin ajamaan tarkistusmoodissa, joten tätä ominaisuutta hyödynnettiin käyttöönottoputkeen yksikkötestausvaiheessa. Koodianalyysiin sen sijaan löytyi valmis Ansible Lint -niminen Python-ohjelma. Sen avulla pelikirjasta oli helppo tarkistaa, että se ei sisältänyt virheitä tai rakenteita, jotka ovat Ansiblen suositusten vastaisia (Ansible, 2020c). Lisäksi ohjelma huomautti koodaustyylistä, minkä tarkoitus on ohjata kaikki kehittäjät samanlaisen koodaustyylin ylläpitämiseen pelikirjoissa.

4.3.3 Verkkoympäristön virtualisointi

Tutkimuksessa käytetty verkkoympäristö toteutettiin virtuaalisilla laitekuvilla (engl. image) tutkijan tietokoneelle. Verkkolaitevalmistaja Aristalta oli mahdollista ladata vEOS-lab -laitekuva, joka on testaukseen käytettävä laboratorioversio heidän virtuaali- ja pilviympäristöissä käytettävästä vEOS-reitittimestä. Lisäksi vEOS vastaa ominaisuuksiltaan Aristan fyysisillä laitteilla käytettävää EOS-ohjelmistoa, joten se sopi siten hyvin tutkimuksen verkkoympäristöön kuvaamaan todellista laiteympäristöä. (Arista, 2017.)

Verkkoympäristön luontiin käytettiin GNS3-ohjelmistoa. Se on avoimen lähdekoodin sovellus, jolla voi luoda erilaisia tietoliikenneverkkoja virtuaalisista laitekuvista. Ohjelmisto koostuu kahdesta komponentista. Graafisella käyttöliittymällä hallitaan verkkotopologiaa, lisätään ja poistetaan laitteita sekä niiden välisiä yhteyksiä. Palvelinkomponentti luo käyttöliittymässä määritellyt virtuaaliset verkkorajapinnat ja -laitteet, sekä huolehtii virtuaalisten laitteiden käynnistämisestä ja pysäyttämisestä. Palvelimena käytettiin paikallista GNS3-virtuaalipalvelinta, jota ajettiin Microsoftin Hyper-V -virtualisointisovelluksella. (Coleman ja muut, 2019.)

GNS3-ohjelmasta otetussa kuvakaappauksessa (ks. kuva 6) näkyy ohjelmalla luotu ja tutkimuksessa käytetty tietoliikenneverkko. Sen rakenteena käytetiin runko ja lehti (engl. spine and leaf) -topologiaa, jossa laitteet sijaitsevat kahdessa kerroksessa. Käyttäjät, palvelimet ja muut verkon palvelut liittyvät verkkoon lehtikerroksen kytkimissä, joiden yhteys runkoon on varmistettu kytkemällä ne kahteen runkokerroksen kytkimeen. Liikennöinti eri lehtikerroksen kytkimissä olevien käyttäjien ja palveluiden välillä tapahtuu siten runkokerroksen kautta. Esimerkiksi kuvassa palomuuri, joka toimii reitittimenä julkisen verkon ja sisäverkon välillä, on kytketty lehtikerroksessa lehti11-kytkimeen, ja mahdollistaa Internet-yhteyden lehti21-kytkimeen liitetylle käyttäjälle.



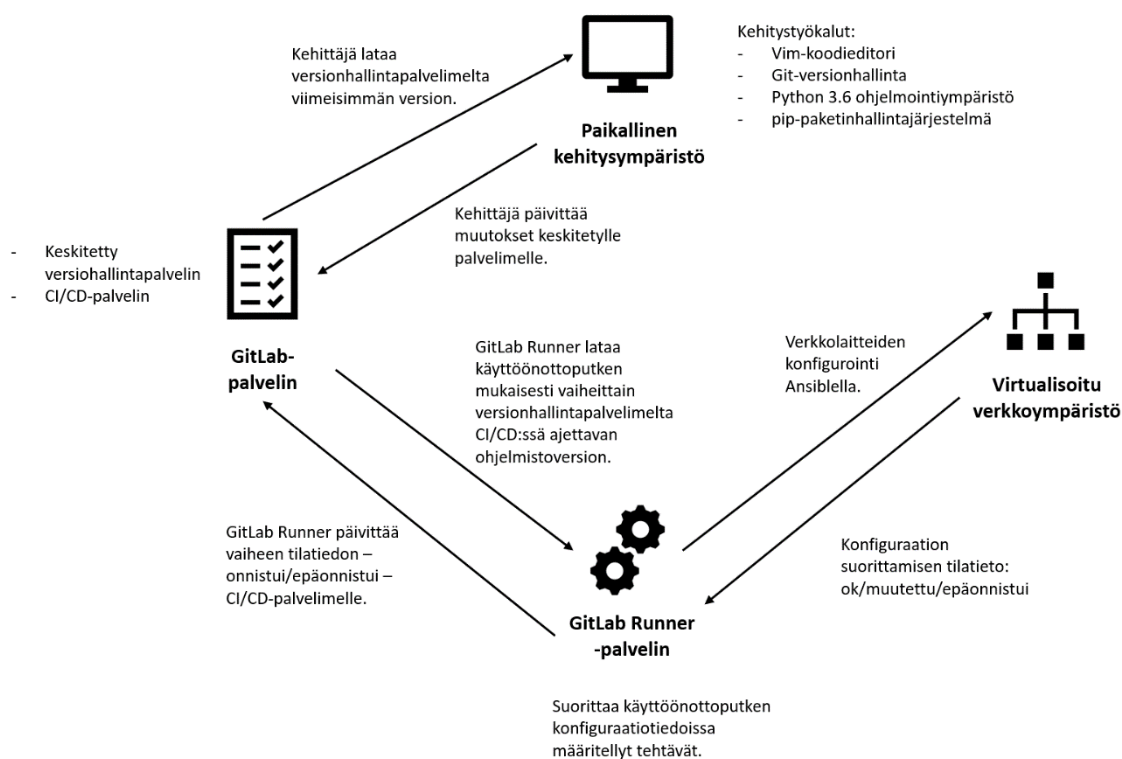
Kuva 6. Tutkimuksessa käytetty testiverkko.

Internet-liikennöinti mahdollistettiin käyttäjälle palveluna, joka toteutettiin VXLAN-tunnelointiteknologialla. Sen avulla lehtikytkimiin liitetyille käyttäjille tai palveluille voidaan konfiguroida pääsy muihin verkkoon kytkettyihin palveluihin. Jotta VXLAN-teknologiaa kyettiin hyödyntämään, piti verkkoon toteuttaa alusverkon (engl. underlay network) ja kerrosverkon (engl. overlay network) konfiguraatiot. Aluskerros konfiguroitiin eBGP-toteutuksella runko- ja lehtikerroksen kytkimien fyysisten liitäntöjen välille, mikä mahdollistaa kahdennetun verkkotopologian toteutuksen. Kerrosverkon konfiguraatiossa käytettiin eBGP-toteutusta verkkojen mainostamiseen, mikä mahdollisti VXLAN-tunneloinnin käyttämisen. (Kelly, 2018.)

4.3.4 Tutkimusympäristö

Alla olevassa kuvassa on esitetty komponentit, joista tutkimusympäristö koostui. Lisäksi kuvaan on mallinnettu komponenttien väliset riippuvuudet. Palvelimien ja verkkoympäristön toteutuksissa hyödynnettiin virtualisointitekniikoita, jolloin ne voitiin toteuttaa

tutkijan omalla tietokoneella, jossa oli käytettävissä 16 Mt keskusmuistia ja prosessorina Intel Core i7-8665U prosessori. Tällä kokoonpanolla koneen resurssit olivat huomattavassa käytössä etenkin keskusmuistin osalta, mutta ne riittivät kuitenkin käyttöönottoputken suorittamiseen hyväksyttävällä nopeudella.



Kuva 7. Tutkimusympäristön komponentit ja niiden väliset riippuvuudet.

Käyttöönottoputken kuvaustiedoston, verkkoympäristön konfiguraation kehitystyö sekä verkkotopologian määrittely tapahtuivat tietokoneen käyttöjärjestelmään asennetussa kehitysympäristössä. GitLab-palvelimella luotiin versionhallintaan ominaisuuksien kehityshaarat, jotka ladattiin paikalliseen ympäristöön kehitettäväksi. Paikallisessa ympäristössä kommitoidut muutokset lähetettiin takaisin GitLab-palvelimelle, jossa ne yhdistettiin päälinjaan.

GitLab-palvelin toimi tutkimusympäristössä keskitettynä Git-versionhallinan säiliönä. Sen käyttöliittymän avulla oli helppo seurata versionhallintaan kommitoituja muutoksia,

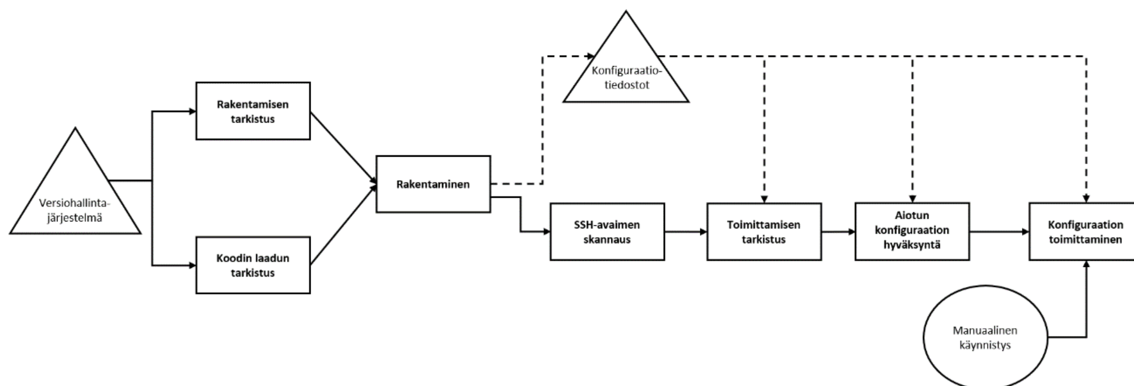
ja verrata tiettyinä aikoina tehtyjä muutoksia keskenään. Uuden kommitoinnin myötä palvelin käynnisti käyttöönottoputken, jonka etenemistä se ohjasi .gitlab-ci.yml -kuvaustiedoston mukaisesti. Palvelimen käyttöliittymällä oli mahdollista seurata käyttöönottoputken etenemisestä, ja se mahdollisti GitLab Runner -palvelimella suoritetun käyttöönottoputken aktiviteettien tarkastelun komento komennolta. Käyttöönottoputken manuaalinen laukaisu onnistui myös käyttöliittymästä.

GitLab Runner -palvelin suoritti käyttöönottoputkessa määritellyt tehtävät. Se kopioi GitLab-palvelimelle kommitoidun version itselleen, ja alkoi suorittamaan .gitlab-ci.yml -tiedostossa kuvattuja tehtäviä, joita olivat mm. verkkolaitteiden konfigurointi Ansiblella.

4.4 Käyttöönottoputken havainnollistaminen

Tässä kappaleessa esittelen suunnittelu- ja kehitysvaiheen tuloksena syntyneen artefaktin, joka on jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin kuvaus käyttöönottoputkena. Toimintamalli on suunniteltu niin, että sitä voidaan hyödyntää verkkolaitteiden konfiguraationhallinnassa tehtyihin muutoksiin versionhallinnassa, ja muuttuneiden konfiguraatioiden toimittamiseen varsinaisille laitteille. Käyttöönottoputki toteutuksessa on myös huomioitu kpl:ssa 4.3 esitetyt, teorian pohjalta kootut jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin tavoitteet.

Käyttöönottoputki on esitetty alla olevassa kuvassa 8. Se on suunniteltu siten, että versionhallinnassa voidaan hyödyntää kehityshaaroja uusien toimintojen toteutukseen. Erillisistä versionhallinnan haaroista huolimatta käyttöönottoputki suoritetaan sekä kehityshaarassa että päälinjassa samoja solmuja ja skriptejä hyödyntäen. Mikäli jokin aktiviteettisolmu epäonnistuu, suoritus keskeytyy, ja versionhallinnan haara on tällöin rikkinäisessä tilassa. Tutkimuksessa käyttöönottoputken tilannetta ja aktiviteettisolmujen tulosten raportointia pystyi seuraamaan GitLab-palvelimella, josta oli luettavissa GitLab Runner -palvelimen ajamien aktiviteettien tilanne ja skriptien tulosteet.



Kuva 8. Tutkimuksessa toteutettu käyttöönottoputki.

Kuvassa on käyttöönottoputken solmuista kerrottu ainoastaan nimet. Solmut sekä niiden tehtävät ja attribuutit on kuvattu tarkemmin alla. Samalla on kerrottu tarkemmin toteutuksessa tehdyistä valinnoista. Lopuksi solmuista on koottu vielä yhteenveto taulukkoon.

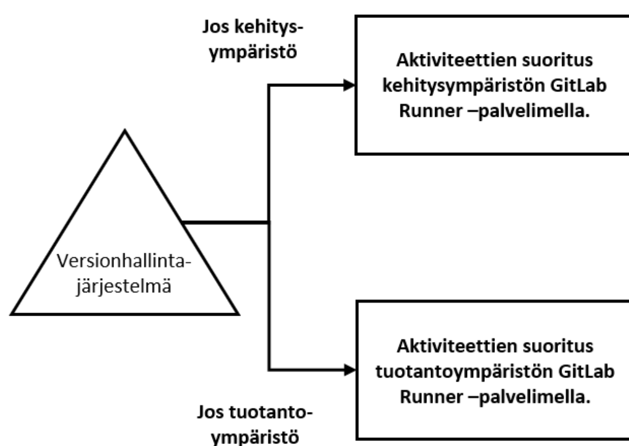
Solmu: Versionhallintajärjestelmä

Päivitys versionhallintaan tapahtuu joko päälinjaan tai uuden toiminnon kehityshaaraan. Päälinjaan ei voi kuitenkaan päivittää suoraan, vaan käyttöönottoputki suoritetaan aina ensin kehityshaarassa. Asiakasympäristöön voi tulla muutospyyntöjä, jotka on toimitettava nopealla ratkaisuajalla esim. kahdessa tunnissa. Siksi päälinjan rikkoontumista on pyritty välttämään ratkaisussa.

Konfiguraation päivityksen yhteydessä ilmoitetaan myös konfiguraatiokohde, joka voi olla joko kehitys- tai tuotantoympäristö. Ilmoitus tapahtuu käyttämällä kumpaankin ympäristöön tarkoitettua tiedostoa, jossa listataan muutoksen kohteena olevien verkkolaitteiden laitenimet. GitLabin käyttöönottoputken konfiguraatiossa tarkistetaan tiedostojen olemassaolo, ja sen mukaan lisätään käyttöönottoputkelle kyseisessä muutoksessa suoritettavat aktiviteettisolmut.

Kehitysympäristössä muutoksen kohteena ovat verkkolaitteet, jotka voivat sijaita joko verkonhallintapalvelun tuottajan tai asiakkaan testiympäristössä. Tuotantoympäristö on

asiakkaan tuotantokäytössä oleva verkkoympäristö. Kehitysympäristö on tarkoitettu pelikirjojen ja Jinja2-mallineiden kehittämiseen. Kun konfiguraationhallintaan ollaan esim. lisäämässä uusia verkkolaitteiden palveluita, tulee ne määritellä konfiguraatiotiedostojen rakenteeseen, kuin myös niistä laitekonfiguraatioita generoiviin mallineisiin. Tuotantoympäristössä sen sijaan muokataan verkon määrityksiä konfiguraatiotiedostoissa, esim. lisäämällä uusia konfiguraatioiden määrittelytiedostoja uusille laitteille, tai muokkaamalla nykyisten laitetiedostojen määrityksiä. Alla olevassa kuvassa on havainnollistettu jakoa ympäristöjen välillä. Ympäristöt on määritelty käyttöönottoputken konfiguraatiossa niin, että kummallakin ympäristöllä on omat aktiviteettisolmunsa. Solmut ovat sisällöltään identtiset, mutta niiden suorittaminen ohjataan eri GitLab Runner -palvelimille. Tuotantoympäristön muutokset voidaan ohjata tehokkaammalle palvelimelle, mikä mahdollistaa käyttöönottoputken suoritusajan optimoinnin asiakasympäristöön.



Kuva 9. Käyttöönottoputken aktiviteettien ohjaus eri palvelimille kohdeympäristöstä riippuen.

Kohdeympäristön valinta vaikuttaa myös siihen, mitkä aktiviteettisolmut suoritetaan versionhallinnan kehityshaarassa. Kun kohteena on kehitysympäristö, suoritetaan koko käyttöönottoputki, kuten alla olevassa kuvassa 10 on esitetty. Näin voidaan turvallisesti varmistaa, että muutokset menevät testeistä läpi, ennen kuin ne yhdistetään päälinjaan. Tämä mahdollistaa konfiguraationhallintajärjestelmän pelikirjojen ja mallineiden kehittämisen niin, että kehittäjä voi varmistua niiden toimivuudesta ennen yhdistämistä

päälinjaan, ja siten välttää päälinjan rikkoontumisen. Mallineissa voi myös hyödyntää koodin piilottamista, jolloin kehityssyklit on mahdollista pitää lyhyenä.

Kuva 10. Käyttöönottoputkessa suoritettavat aktiviteetit, kun kommitoidaan kehityshaaraan.

Tuotantoympäristöön tehdylle muutokselle sen sijaan suoritetaan yllä olevan kuvan mukaisesti ainoastaan *rakentamisen tarkistus*, *koodin laatu* sekä *rakentaminen* aktiviteettisolmut. Nämä kolme solmua suoritetaan paikallisesti GitLab Runner -palvelimella, jolloin niiden suoritusaika on lyhyt. Niillä on helppo varmistaa, että kommitointivaiheen testit tulevat tehdyksi mikäli käy niin, että konfiguraatiotietojen päivittäjä ei ole tehnyt testejä omassa kehitysympäristössään. Koska tuotantoympäristön muutokset on suunniteltu verkon konfiguraatioarvojen lisäyksiksi, poistoiksi ja muutoksiksi konfiguraatioiden määrittelytiedostoissa, paikallisesti ajettavat testit paljastavat nopeasti, mikäli tiedot on täytetty virheellisesti (esim. malline ei pysty luomaan konfiguraatiotiedostoa).

Solmu: Rakentamisen tarkistus

Konfiguraationhallintajärjestelmä mahdollistaa pelikirjojen suorittamisen tarkistusmoodissa, jossa Ansible suorittaa pelikirjan, ilmoittaa mitä se olisi muutoksessa tehnyt, mutta

jättääkin varsinaisen muutoksen tekemättä. Tarkistusmoodilla voidaan siten nopeasti todeta, ovatko pelikirjat Ansiblen kannalta toimivia. Koska pelikirja käyttää Jinja2-mallineita konfiguraatioiden luomiseen, myös niiden syntaksin oikeellisuus voidaan todeta suorituksen yhteydessä.

Solmu: Koodin laatu

Pelikirjojen laatu tarkistetaan ansible-lint -ohjelmalla. Se tunnistaa mikäli pelikirjoissa on käytetty Ansiblen moduuleja, jotka ovat vanhentumassa tai jo vanhentuneet ja siten Ansiblen suositusten vastaisia (Ansible, 2020). Ladatessaan pelikirjaa työkalu myös tarkistaa, että tiedosto noudattaa YAML-kuvauskielen syntaksia.

Solmut: Rakentaminen ja konfiguraatiotiedostot

Verkkolaitteiden konfiguraatiot luodaan Jinja2-mallineiden avulla. Konfiguraatiotiedostot tallennetaan GitLab-palvelimelle artefakteiksi, joita hyödynnetään syötteinä myöhemmissä aktiviteeteissa saman käyttöönottoputken suorituksen aikana.

Solmu: SSH-avaimen skannaus

Konfiguraationhallintatyökalu liikennöi verkkolaitteisiin SSH-protokollalla (Secure Shell), joka käyttää laitteiden julkista avainta salatun yhteyden muodostamiseen. Koska käyttöönottoputki suoritetaan GitLab Runner -palvelimella, tulee verkkolaitteiden julkisen avaimen tallennus suorittaa automaattisesti pelikirjan skriptillä. Tallennusta ohjataan kehitys- ja tuotantoympäristöä varten luoduilla tiedostoilla, joissa listataan avaimen tallennuksen tarvitsevien laitteiden laitenimet. Ohjaus tapahtuu siten samalla tavalla kuin käyttöönottoputken kohdeympäristön valinnassa. Mikäli ohjaustiedosto puuttuu, aktiviteettisolmua ei suoriteta. Kehittäjä määrittää tallennuksen tehtäväksi, kun tiedetään että GitLab Runner -palvelin ottaa yhteyden laitteeseen ensimmäisen kerran. Yleensä tämä tapahtuu, kun verkkoon lisätään uusi laite tai vikaantunut laite korvataan uudella.

Solmu: Toimittamisen tarkistus

Aktiviteetti hyödyntää samaa tarkistusmoodia, kuin mitä käytettiin *rakentamisen tarkistus* -aktiviteetissa. Tässä kohtaa käyttöönottoputkea kaikkia pelikirjojen tehtävien komentoja ei suoriteta paikallisesti palvelimella, vaan osa niistä ajetaan kohdelaitteessa. Aktiviteetissa tarkistetaan ensin ping-komennolla, että palvelin saa yhteyden kohdelaitteeseen. Tämän jälkeen konfiguraationhallintatyökalu testaa pelikirjan toimivuuden.

Solmu: Testaa aiottu konfiguraatio

Aktiviteetti testaa jälleen aluksi yhteyden toimivuuden kohdelaitteeseen. Seuraavaksi se kopioi laitteelle sille tarkoitetun konfiguraation. Tässä vaiheessa pelikirja hyödyntää Aristan EOS:n ominaisuutta luoda konfiguraatiomuutokset erilliseen istuntoon, joka ei vaikuta ajossa olevaan laitekonfiguraatioon. Näin pystytään toteamaan, että kohdelaite hyväksyy konfiguraation, eli rakennetussa konfiguraatiossa ei ole syntaksivirheitä tai virheellisiä komentojen argumentteja. Lisäksi aktiviteetti ajaa lopuksi vertailun ajossa olevan ja aiotun konfiguraation välillä. Konfiguraatioiden mahdolliset erot se raportoi konfiguraationhallintajärjestelmän tulosteessa. Näin muutoksen tekijä pystyy aktiviteetin valmistuttua vielä tarkistamaan, onko uusi konfiguraatio mahdollisesti ylikirjoittamassa laitteella olevaa konfiguraatiota, mitä ei ollut tarkoitus alun perin tehdä. Koska konfiguraation toimittaminen käynnistetään manuaalisesti tämän aktiviteetin jälkeen, jää tekijälle mahdollisuus selvittää syy konfiguraatioiden eroavuuteen. Mikäli tämä tieto ei ole nopeasti saatavilla, joutuu muutoksen peruuttamaan päälinjaan, koska se estää seuraavien päivitysten toimittamisen käynnistämisen. Tilanne tulee eteen, mikäli laitteelle on tehty konfiguraatiomuutoksia ohi versionhallinnan ja käyttöönottoputken.

Solmu: Toimita konfiguraatio

Käyttöönottoputken viimeisessä aktiviteetissa testataan jälleen ensimmäisenä yhteyden toimivuus kohdelaitteelle. Tämän jälkeen aktiviteetin työjärjestys on seuraavanlainen: tee varmuuskopio laitteella ajossa olevasta konfiguraatiosta, kopioi konfiguraatiotiedosto laitteen ajossa olevaksi konfiguraatioksi, kopioi ajossa oleva konfiguraatio laitteen käynnistyskonfiguraatioksi. Kun uusi konfiguraatio on paikoillaan, tarkistetaan lopuksi sekä sen toimivuus että se sisällöltään täyttää halutut vaatimukset. Tarkistukset suorittavalla pelikirjalla todetaan, että ainoastaan sallitut käyttäjät ovat konfiguroituna, alus- ja kerrosverkot ovat muodostaneet naapurussuhteet, ja että laitteiden valvonta on mahdollista SNMP-protokollan (Simple Network Management Protocol) avulla. Mikäli tarkistus epäonnistuu jossain kohdassa, tehdään konfiguraation palautus aktiviteetin alussa tehdystä varmuuskopiosta, ja käyttöönottoputki epäonnistuu.

Yhteenveto käyttöönottoputken solmuista

Alla olevaan taulukkoon on koottu vielä yhteenvetona käyttöönottoputken eri solmut ja niiden tyypit. Solmuista on kerrottu lyhyt kuvaus sekä missä versiohallinnan haarassa ja kohdeympäristössä kyseinen solmu suoritetaan.

Taulukko 3. Käyttöönottoputken solmut, tyypit ja kuvaukset.

Solmu	**Solmun tyyppi**	**Kuvaus**
Versionhallinta-järjestelmä	Syöte	Kommitointi versionhallintajärjestelmään käynnistää käyttöönottoputken suorituksen GitLab-palvelimelta.
Rakentamisen tarkistus	Aktiviteetti	Tarkistetaan, että konfiguraatioiden rakentamiseen määritellyt Ansiblen pelikirja sekä Jinja2-mallineet ovat toimivia. Suoritetaan • versionhallinnan kehityshaarassa kehitys- tuotantoympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.

Koodin laadun tarkistus	Aktiviteetti	Tarkistetaan ansible-lint -ohjelmalla, että pelikirjojen moduulit ovat ajantasaisia ja YAML-syntaksi kunnossa. Suoritetaan • versionhallinnan kehityshaarassa kehitys- tuotantoympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.
Rakentaminen	Aktiviteetti	Luodaan konfiguraatiotiedostot, jotka toimitetaan myöhäisemmässä aktiviteetissa verkkolaitteille. Tallennetaan artefaktiksi. Suoritetaan • versionhallinnan kehityshaarassa kehitys- tuotantoympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.
Konfiguraatio-tiedostot	Syöte	Artefakteina GitLab-palvelimelle tallennetut konfiguraatiot. Tallennetaan • versionhallinnan kehityshaarassa kehitys- tuotantoympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.
SSH-avaimen skannaus	Aktiviteetti	Tarvitaan verkkolaitteen julkisen SSH-avaimen tallennukseen ja määritellään suoritettavaksi silloin, kun GitLab Runner -palvelin ottaa ensimmäisen kerran yhteyden laitteeseen. Suoritetaan • versionhallinnan kehityshaarassa kehitysympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.
Toimittamisen tarkistus	Aktiviteetti	Tarkistetaan, että konfiguraatioiden toimittamiseen määritelty Ansiblen pelikirja on toimiva.

		Suoritetaan • versionhallinnan kehityshaarassa kehitysympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.
Aiotun konfiguraation hyväksyntä	Aktiviteetti	Konfiguraatio toimitetaan verkkolaitteelle erilliseen istuntoon, jossa voidaan testata, että laite hyväksyy sille tarkoitetun konfiguraation. Tulos raportoidaan ja se on tarkistettavissa GitLab-palvelimella. Suoritetaan • versionhallinnan kehityshaarassa kehitysympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.
Manuaalinen käynnistys	Ulkoinen laukaisin	Konfiguraation toimittaminen -aktiviteetin käynnistys GitLab-palvelimelta.
Konfiguraation toimittaminen	Aktiviteetti	Varmuuskopioi laitteen ajossa olevan konfiguraation GitLab Runner -palvelimelle, minkä jälkeen kopioi uuden, rakennetun konfiguraation laitteelle. Tekee toimituksen tarkistuksen, jossa todennetaan että palvelut toimivat halutulla tavalla. Mikäli tarkistus epäonnistuu, palautetaan varmuuskopioitu konfiguraatio laitteelle. Suoritetaan • versionhallinnan kehityshaarassa kehitysympäristöön. • versionhallinnan päälinjassa kehitys- ja tuotantoympäristöön.

5 Arviointi

Kappaleessa 4.3 listasin jatkuvan integroinnin ja jatkuvan toimittamisen teorian pohjalta kootut tavoitteet käyttöönottoputkelle. Suunnittelusykleissä on havainnoitu, miten käyttöönottoputki voidaan toteuttaa siten, että sillä on mahdollista tehdä verkonhallintaa tavoitteiden mukaisesti. Alla olevissa kappaleissa olen aiheittain arvioinut, miten tavoitteet on käyttöönottoputkessa saavutettu, tai miksi jokin tavoite on jätetty toteuttamatta. Tavoitteet ovat omina kappaleinaan siihen liittyvän aiheen alla.

Käyttöönottoputki esiteltiin alfa-yritykselle simuloimalla tutkimuksessa käytetyn verkon provisiointi aloituskonfiguraatiosta tuotantovalmiiksi, sekä lisäämällä SNMP-konfiguraatio uuden palvelun kehityksenä konfiguraationhallintaan ja testiverkkoon. Simulointia käytettiin siten ratkaisun testinä ympäristössä, jossa tutkimuksen ongelma oli alun perin tunnistettu. Lisäksi käytiin läpi alla olevissa kappaleissa käsitellyt jatkuvan integroinnin ja jatkuvan toimittamisen havainnot käyttöönottoputken suunnittelussa. Alfa-yrityksen palvelutuotannon organisaatiosta simulaatioon osallistuivat palvelutuotannon esimies, jatkuvan palvelun ylläpidon esimies ja tiimien vetäjät, palvelutuotantoympäristön tiimin vetäjä sekä teknologiavastaava. Seuraavassa olen referoinut osallistujien artefaktista antamia kommentteja sekä heidän esittämiä kysymyksiä.

Yleisesti jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia pidettiin toimivana, ja käyttöönottoputki esitti hyvin, miten sitä voidaan hyödyntää verkkolaitteiden hallintaan.

> *Käyttöönottoputki on selkeä malli, joka itsessään ohjaa käytännön tekemistä oikeaan suuntaan. (Teknologiavastaava)*

> *Käyttöönotto vaatii kuitenkin myös hallintamallin ja prosessien luomista, jotta tiimit pystyvät hyödyntämään toimintamallia organisaatiolle optimaalisella tavalla. (Palvelutuotannon esimies)*

Toimintamalliin oleellisesti sisältyvä runsas testaaminen huomioitiin toimituksen tarkistusvaiheessa. Toisaalta, huomiota kiinnitettiin heti myös siihen, että testit tulee myös suunnitella ja määritellä pelikirjoihin.

> *Oli hyvä, että uusille palveluille määritellään heti palvelun suunnitteluvaiheessa myös tarkistukset. Toisaalta, muutokset voivat kestää pidempään, jos pitää kirjoittaa myös testit. Kun tämän pitää mielessä, CI/CD vaikuttaa hyvältä. (Teknologiavastaava)*

Konfiguraationhallintajärjestelmän toteutuksesta ja hyödyntämisestä tutkimuksessa käytiin sen ominaisuuksiin liittyviä keskusteluja. Tämä oli luonnollista, koska se on lähempänä käytännön tekemistä verkkolaitteiden konfiguroimisessa kuin jatkuva integroiminen ja jatkuva toimittaminen. Järjestelmän hyödyntäminen alfa-yrityksen toimintaympäristössä on kuitenkin itsessään laaja aihe. Tässä yhteydessä todettiin, että tutkimuksessa toteutettiin yhdenlainen ratkaisu konfiguraatioiden määrittelytiedostoille, joita sitten hyödynnettiin Jinja2-mallineissa. Konfiguraatiohallintajärjestelmän hyödyntäminen jatkuvassa integroinnissa ja jatkuvassa toimittamisessa herätti kahdenlaisia mielipiteitä.

- *Teknologiavastaava: Kannattaisi pyrkiä kuvaamaan koko asiakasverkko, jolloin se saadaan kokonaisuudessaan käyttöönottoputken toimintamalliin ja sitä voidaan hyödyntää kaikissa konfiguraatiomuutoksissa.*
- *Jatkuvan palvelun ylläpidon esimies: Näin laajaan verkkojen kuvaamiseen asiakaskohtaisesti ei kannata ryhtyä, vaan olisi parempi keskittyä verkon palveluihin, joissa useimmiten tehdään konfiguraatiomuutoksia.*

Automaation hyödyntäminen tuo mukanaan myös uusia lähestymistapoja konfigurointiin, kun konfiguraatiota ei tehdäkään suoraan laitteella.

> *Miten tunnistaa konflikti nykyisessä ja uudessa konfiguraatiossa? (Teknologiavastaava)*

Käyttöönottoputkessa tämä oli ratkaistu aiotun konfiguraation testauksessa, jonka tuloksesta muutoksen tekijä näkee hyväksyykö laite konfiguraation, ja miten uusi konfiguraatio eroaa nykyisestä laitteella käytössä olevasta konfiguraatiosta. Tuloksen perusteella

muutoksen tekijä voi manuaalisesti käynnistää konfiguraation toimittamisen. Tähän liittyen myös palautusvaihtoehdot mietityttivät.

> *Testeissä kaikki näyttää hyvältä, mutta tuotannossa muutoksen jälkeen jokin palvelu ei toimikaan. Voidaanko rakentaa palautusmetodeja, olettaen että yhteydet laitteisiin ovat edelleen olemassa? (Teknologiavastaava)*

Tutkimuksessa oli palautusmenetelmänä hyödynnetty konfiguraation tallentamista GitLab Runner -palvelimelle ennen muutosta, ja mikäli toimituksen tarkistustesti epäonnistuvat, konfiguraatio palautetaan automaattisesti tallennettuun versioon. Lisäksi konfiguraation määrittelytiedostossa oli mahdollista määritellä, mikäli jokin testeistä halutaan jättää suorittamatta. Tällainen tilanne voi tulla vastaan esim. muutoksessa, jossa tiedetään että jokin palvelu ei kuulukaan toimia, ennen kuin muutos on tehty kokonaisuudessaan valmiiksi.

5.1 Nopea palaute

5.1.1 Palaute testauksesta CI:ssä ja CD:ssä

Jotta kehittäjä saa testauksesta palautteen mahdollisimman tehokkaasti, suosittavat sekä Humble ja Farley (2010) että Ståhl ja Bosch (2014a) testien sijoittamista käyttöönottoputkella siten, että nopeammat testit ovat putken alussa ja hitaammat lopussa. Tätä suositusta voitiin hyödyntää myös tutkimuksessa käyttöönottoputkelle, ja siinä käytetty konfiguraationhallintajärjestelmä mahdollistaa nopeat testaukset käyttöönottoputken alussa kahdella tavalla. Pelikirjojen laadun tarkistukseen on tehty ansible-lint -työkalu, joka varoittaa mahdollisista parannuksista pelikirjojen kuvauksissa. Lisäksi Ansiblella on mahdollista suorittaa pelikirjat tarkistusmoodissa, millä voi todentaa, että pelikirjojen tehtävien käyttämät komennot ja työkalut (esim. tutkimuksessa Jinja2-mallineet) ovat toimivia. Kaikkia pelikirjoja ei kuitenkaan testattu kerralla, vaan tarkistusmoodin testit jaettiin kahteen vaiheeseen integroinnin ja toimittamisen mukaisesti. Käyttöönottoputken alussa, kun muutokset integroidaan keskitettyyn versionhallintaan, suoritetaan

konfiguraatioiden rakentamiseen liittyvän pelikirjan tarkistus. Testaus tapahtuu paikallisesti GitLab Runner -palvelimella. Lopuksi konfiguraationhallintajärjestelmällä siirretään konfiguraatiot kohdelaitteisiin. Toimittamisen pelikirjan testaus tarkistusmoodissa tapahtuukin myöhäisemmässä vaiheessa, koska pelikirjan tehtävät ottavat yhteyden laitteeseen. Testaus tehdään jokaiselle muutoksen kohteena olevalle laitteelle, jolloin varmistutaan että varsinainen konfiguraation toimittaminen niihin onnistuu pelikirjaa käyttämällä.

Suoritusaikatavoitetta käsittelevässä kappaleessa olen testannut käyttöönottoputken aktiviteettien suoritusaikoja. Sen sisältämästä taulukosta nähdään, miten nopeat testit sijoittuvat alkuun ja hitaammat loppuun.

5.1.2 Jatkuvalla toimittamisella saavutettava nopea kierrosaika ja asiakaspalaute

Muutosten tekeminen verkon konfiguraation onnistuu nopeasti konfiguraation määrittelytiedostoa muokkaamalla. Etenkin usealle laitteelle saman muutoksen tekeminen on tehokasta, kun jokaiseen laitteeseen ei tarvitse ottaa yhteyttä ja tehdä muutosta manuaalisesti. Toisaalta, konfiguraationhallintajärjestelmää varten määritellyn konfiguraatiotiedoston rakenne vaikuttaa siihen, miten helposti käyttäjä pystyy muokkaamaan konfiguraatiota siten, että syntaksi säilyy oikeassa muodossa. Tämä korostuu kun konfiguraationhallintaa käytetään useille asiakkaille erilaisissa ympäristöissä. Tutkimuksessa ei selvitetty millainen olisi hyvä rakenne, jolla käyttäjän olisi helppo muokata monimutkaisiakin konfiguraatioita, vaan toteutus kohdistettiin käytettyyn verkkoympäristöön ja teknologiaan.

Verkon konfiguraation muuttaminen ei yleisesti ole aikakriittinen toimenpide, eikä yllä kuvattu muutoksen tekeminen nopeudeltaan eroa yksittäisen laitteen manuaalisesta konfiguroinnista. Kun muutoksia tehdään usealle laitteelle, eroa alkaa muodostua jatkuvan integroinnin ja jatkuvan toimittamisen eduksi. Chen (2017), Leppänen ja muut (2015) sekä Ohlsson ja muut (2012) havaitsivat, että jatkuvan toimittamisen valmiiden

toteutusten lyhyt kierrosaika kehityksestä tuotantoon lisäsi nopeutti myös asiakaspalautteen saamista. Tietoliikenneverkon yksittäisissä verkkolaitteen konfiguraatiomuutoksissa tällä ei ole suurempaa merkitystä pl. laaja, useaan laitteeseen kohdistuva muutos. Sen sijaan koko asiakasverkon näkökulmasta jatkuvan integroinnin ja jatkuvan toimittamisen hyödyntäminen sen kierrosaikaa parantavien ominaisuuksien osalta tarkoittaisi, että verkko on jatkuvassa muutoksessa, jossa laitemalleja päivitetään tasaisesti uudempiin versioihin. Tällöin sekä asiakkaan että verkonhallintapalvelun tuottajan tulee kyetä luottamaan käyttöönottoputkeen niin, että laitepäivityksiä voidaan tehdään verkkoon jatkuvasti, jopa ilman muutosikkunoita. Näin asiakas voi jakaa verkon päivityksestä aiheutuvat kustannukset usealle vuodelle, sen sijaan että tehtäisiin koko verkon päivitys kerrallaan muutaman vuoden välein. Malli vaatii pitkän toimitussuhteen palvelun tuottajan ja asiakkaan välille, jotta muutoksessa oleva vanha ympäristö saadaan kuvattua konfiguraationhallintaan riittävällä tarkkuudella, mikä mahdollistaa täsmällisesti tehtävät päivitykset eri laiteversioiden ja teknologioidenkin välillä.

5.1.3 Uusilla palveluilla oltava lyhyet kehityssyklit

Tutkimuksessa käytetyssä verkkotopologiassa palvelut voidaan rakentaa siten, että niiden tekeminen jaetaan pienemmiksi kokonaisuuksiksi, esimerkiksi alusverkon (engl. underlay network) ja kerrosverkon (engl. overlay network) komponenteiksi. Näin palveluiden suunnittelu ja toteutus konfiguraationhallinnan kannalta tapahtuu lyhyissä sykleissä. Lisäksi oleelliset tehtävät – konfiguraatiotiedoston rakenteen, Jinja2-mallineiden sekä komponenttien toimivuuden varmistavien testien suunnittelu ja toteutus – voidaan tarvittaessa jakaa omiksi kehitysvaiheiksi.

5.1.4 Työnkulun on oltava välitön

Kehittäjän tai konfiguraation muutoksen tekijän on mahdollista kommitoida muutoksensa versionhallintaan aina halutessaan, kuten Ståhl ja Bosch (2014b) ovat ohjeistaneet.

Aktiviteettien suoritus käynnistyy automaattisesti ja viimeisen aktiviteetin tekijä voi käynnistää itse. Myös kehityshaaran yhdistäminen päälinjaan voi kehittäjä tehdä itsenäisesti, mutta tällöin on varmistuttava, että päälinja ei ole rikkinäisessä tilassa.

5.1.5 Työnkulun on oltava läpinäkyvä

Ståhl ja Bosch (2014b) ovat ohjeistaneet myös, että käyttöönottoputken tulee olla selkeä ja kaikkien ymmärrettävissä. Tutkimuksessa käytetyssä jatkuvan integroinnin järjestelmässä kaikilla projektin jäsenillä on pääsy versionhallinnan säiliöön, johon jatkuvan integroinnin järjestelmä tallentaa myös käyttöönottoputken konfiguraatiotiedoston. Työnkulku on selkeästi jaettu kehitys- ja tuotantoympäristöön kohdistuviin muutoksiin.

5.1.6 Tiimin yhteistyöllä rakentama käyttöönottoputki

Chen (2017) sekä Humble ja Farley (2010) mukaan käyttöönottoputki kannattaa rakentaa valmiiksi jatkuvan toimittamisen menetelmien mukaisesti projektitiimin toimesta projektin edetessä. Palvelutuotanto-organisaatiossa, jossa useita erilaisia asiakasverkkoja konfiguroidaan jatkuvan integroinnin ja jatkuvan toimittamisen menetelmillä, voi konfiguraatiomuutoksia tehdä henkilöt, jotka eivät ole olleet mukana alkuperäisen verkkoympäristön rakentamisprojektissa. Tällöin käyttöönottoputki kannattaa vakioida niin, että sen toiminta on pääasiassa samanlainen ympäristöstä riippumatta. Jokainen uusi projekti hyödyntää siten valmista käyttöönottoputken mallinetta, jonka rakentumiseen koko tiimi on parhaassa tapauksessa ajan kanssa päässyt vaikuttamaan. Näin tiimi myös ymmärtää sen toimintatavan, ja voi menestyksellä hyödyntää jatkuvan toimittamisen toimintamallia, kuten Leppänen ja muut (2015) ovat todenneet.

5.2 Versionhallinta

5.2.1 Käytä vain päälinjaa

Versionhallinnassa käytetään muutoksien tekemiseen kehityshaaraa. Muutoksia on suunniteltu olevan kahdenlaisia, ja käyttöönottoputken suoritustapa riippuu siitä, kummanko tyyppistä muutosta ollaan tekemässä. Kun konfiguraationhallintaan ollaan luomassa uutta palvelua, tehdään muutoksia konfiguraation määrittelytiedoston rakenteeseen, Jinja2-mallinteisiin ja toimivuuden varmistaviin testeihin. Tällöin kehityshaarassa suoritetaan koko käyttöönottoputki, ja kehittäjän kohdelaitteina ovat testiympäristössä olevat verkkolaitteet. Muutoksen kohdistuessa olemassa olevan palvelun konfiguraatiotietoihin, suoritetaan kehityshaarassa ainoastaan paikallisesti GitLab Runner -palvelimella ajettavat testit. Näin muutoksen tekijöiden on helppo varmistua, että kaikki kyseiseen verkkoympäristöön suunnitellut käyttöönottoputken testit tulevat varmasti tehtyä ennen yhdistämistä päälinjaan, jolloin varmuus päälinjan toimivuudesta yhdistämisen jälkeen kasvaa. Kehityshaaran käyttö on Fowlerin (2006), Humblen ja Farleyn (2010) sekä Ståhlin ja Boschin (2014a) suosituksen vastaista. Ratkaisussa on kuitenkin vältetty päälinjan rikkoontumista. Olemassa olevien palveluiden muutospyyntöjä voi joutua toimittamaan saman päivän aikana, jolloin työn aloittaminen ehjään päälinjaan on varmempaa. Muutoksen voi toteuttaa henkilö, joka ei alun perin ole työskennellyt kyseisen verkkoympäristön kanssa. Palvelutuotannon sujuvaa toimivuutta olemassa olevaa verkkoympäristöä vasten on siten painotettu, ja tiimi joka työstää kokonaan uutta palvelua, pystyy toimimaan rinnalla uuden kehitystyön kanssa.

5.2.2 Hyödynnä kommunikointivälineenä

Koska konfiguraationhallintajärjestelmä käyttää tekstitiedostoja pelikirjoille, mallineille ja konfiguraatiotiedostoille, on versionhallinnasta helppo nähdä muutoksen tekijä, sisältö ja ajankohta. Integraatiojärjestelmän käyttöliittymässä eri ajankohtina tehtyjä muutoksia on visuaalisesti helppo vertailla asettamalla eri versioiden tiedostot vierekkäin samaan näkymään. Järjestelmä helpottaakin huomattavasti versionhallinnan

hyödyntämistä kommunikointiin ja nopean palautteen saamiseen, kuten Fowler (2006) sekä Humble ja Farley (2010) suosittavat.

5.2.3 Päälinjan on pysyttävä eheänä

Kuten aiemmin on todettu, päälinjan eheyteen on kehityshaaroja käyttämällä kiinnitetty enemmän huomiota kuin mitä teoriassa on suositeltu. Siten rikkoontuneen päälinjan korjaamisen tärkeys on yhtenevä Fowlerin (2006) sekä Humblen ja Farleyn (2010) ohjeistuksen kanssa. Ståhl ja Bosch (2014) havaitsivat alan kirjallisuudessa, että mielipide päälinjan eheyden kriittisyydestä kuitenkin vaihtelee, eikä rikkoontumista kannata pelätä. Kriittisyys riippuu osaksi varmasti myös työn alla olevasta projektista. Usean eri verkkoympäristön ja tiimin kanssa toimittaessa sääntö päälinjan eheydelle kannattaa olla selkeän yksiselitteinen.

5.3 Testaus

5.3.1 Suoritusaika voi olla muutamia minuutteja

Jatkuvan integroinnin testeille on suositeltu maksimissaan kymmenen minuutin suoritusaikaa, jotta kehittäjä saa riittävällä nopeudella tiedon koodinsa integroinnin onnistumisesta (Fowler, 2006; Humble &Farley, 2010). Verkkolaitteiden konfiguraatiossa muutoksen tekijä saa varmuuden muutoksen onnistumisesta kun käyttöönottoputken viimeinenkin aktiviteetti on suoritettu. Siksi koko käyttöönottoputken suoritusaika olisi hyvä olla alle suositellun ajan. Tuotantoympäristöön muutosta tekevä asiantuntija tekee muutoksen mieluummin suoraan laitteelle, mikäli käyttöönottoputken suoritusaika on jatkuvasti pidempi kuin mitä muutoksen manuaaliseen tekemiseen kuluisi. Toisaalta on hyvä muistaa, että konfiguraationhallintajärjestelmällä muutoksen tekeminen useampaan laitteeseen on huomattavasti nopeampaa kuin manuaalisesti tehtynä. Jatkuvan integroinnin järjestelmän nopeuteen vaikuttaa millaisille palvelimille se on asennettu. Myös

konfiguraationhallintajärjestelmän nopeus tehdä muutoksia yksittäiseen tai useampaan laitteeseen vaikuttaa aktiviteettien suoritusten kestoon.

Alla olevaan taulukkoon on koottu tulokset kolmesta käyttöönottoputken testisuorituksesta. Tuloksilla on tarkoitus havainnollistaa, miten pitkään koko käyttöönottoputken suoritukseen kuluu tutkimusympäristössä, sekä eri aktiviteettien välisiä suoritusnopeuksia. Ensimmäinen testi tehtiin uudelleen käynnistetyllä tietokoneella, jossa käyttöön otettiin ainoastaan testiympäristö ja sen käyttämiseen tarvittavat sovellukset. Seuraavat testit suoritettiin samassa, käynnissä olleessa testiympäristössä. Verkkolaitteiden konfiguraatiot palautettiin aloituskonfiguraatioon ja SSH-avaimet poistettiin GitLab Runner -palvelimelta. Kohteena testeissä oli tuotantoympäristö, ja niissä tehtiin koko verkkoympäristön provisiointi tuotantovalmiiksi. Integrointijärjestelmässä on mahdollista hyödyntää välimuistia aktiviteettisolmuissa tarvittavien sovelluspakettien tallentamiseen GitLab Runner -palvelimelle valmiiksi, jolloin niitä ei tarvitse ladata ja asentaa jokaisessa solmussa Internetistä erikseen. Tätä ominaisuutta hyödynnettiin käyttöönottoputkessa. Taulukosta nähdään, että tuotantoon muutosta tehtäessä suoritusaika testiympäristössä jää kokonaisuudessaan alle suositellun integraatioajan. Tähän vaikuttaa konfiguroitavien laitteiden määrä, mutta testiympäristön tuloksissa on olennaista, että suoritusaika jää lyhyeksi pienellä laitemäärällä. Lisäksi tuloksista nähdään, miten paikallisesti GitLab Runner -palvelimella suoritetut aktiviteettisolmut ovat selvästi nopeampia, kuin verkkolaitteisiin yhteyden ottavat solmut. Käyttöönottoputken alkupäässä ne antavat nopean palautteen muutoksen tekijälle.

Taulukko 4. Käyttöönottoputkelle testatut suoritusajat.

Aktiviteettisolmu	Testi 1	Testi 2	Testi 3
Rakentamisen tarkistus	23 s	27 s	23 s
Koodin laadun tarkistus	24 s	26 s	22 s
Rakentaminen	23 s	23 s	22 s
Yhteensä (kehityshaara)	**1 min 10 s**	**1 min 16 s**	**1 min 7 s**
Rakentamisen tarkistus	25 s	26 s	29 s

Koodin laadun tarkistus	32 s	22 s	23 s
Rakentaminen	24 s	22 s	22 s
SSH-avaimen skannaus	44 s	45 s	44 s
Toimittamisen tarkistus	1 min 6 s	52 s	54 s
Aiotun konfiguraation hyväksyntä	52 s	47 s	50 s
Konfiguraation toimittaminen	2 min 51 s	2 min 36 s	2 min 34 s
Yhteensä (päälinja)	**6 min 54 s**	**6 min 10 s**	**6 min 16 s**
YHTEENSÄ (kehityshaara + päälinja)	**8 min 4 s**	**7 min 26 s**	**7 min 23 s**

5.3.2 Yksikkötestit

Konfiguraationhallinnassa ei käytetty yksikkötestausta samalla tavalla kuin sovellusten ohjelmoinnissa (Humble & Farley, 2010). Sen sijaan hyödynnettiin Ansiblen mahdollistamaa pelikirjan suorittamista tarkistusmoodissa, jossa Ansible käy läpi pelikirjan tehtävät ja ilmoittaa mitä se olisi tehtävässä tehnyt, jättäen kuitenkin varsinaisen muutoksen tekemättä. Ominaisuuden avulla on siten helppo tarkistaa, että pelikirjan tehtävät ovat toimivia eivätkä sisällä syntaksivirheitä. Tarkistusmoodi ei kuitenkaan toimi tilanteissa, joissa pelikirjan tehtävä on riippuvainen aiemman tehtävän tuloksista, joten sen hyödyntämismahdollisuus on riippuvainen myös pelikirjan käyttötarkoituksesta. Tutkimuksen käyttöönottoputkessa keskeisimmät pelikirjat, konfiguraatioiden rakentaminen ja toimittaminen, pystyttiin testaamaan tällä toiminnolla.

Koska tarkistusmoodissa Ansible suorittaa pelikirjaa tehtäviä, testit jaettiin eri aktiviteettisolmuihin. Paikallisesti palvelimella suoritettava konfiguraation rakentamisen pelikirja testattiin heti käyttöönottoputken alussa. Konfiguraation toimittaminen testattiin ensimmäisenä verkkolaitteisiin yhteyden ottavista, konfiguraation toimittamiseen liittyvistä aktiviteeteista.

5.3.3 Komponenttitestit

Komponenttitestausta ei käyttöönottoputkessa tehdä, koska varsinaisia testattavia komponentteja ei konfiguraationhallinnasta tunnistettu. Valmistajien tekemät verkkolaitteiden ohjelmistoversiot voidaan kuitenkin ajatella komponentteina, jotka vaikuttavat konfiguraatioon. Ohjelmistopäivitys tehdään yleensä silloin, kun uusi versio korjaa tietoturva-aukkoja, nykyisten palveluiden ohjelmistovirheitä, tai mahdollistaa uusien palveluiden käyttöönottamisen. Ennen uuden version käyttöönottoa tuotannossa se voidaan asentaa testiympäristöön. Näin kehityshaarassa voidaan todentaa, että nykyisillä malleilla luodut konfiguraatiot toimivat myös uudessa versiossa, ja että ne läpäisevät hyväksyntätestit.

5.3.4 Hyväksyntätestit

Kun verkkolaitteelle syöttää konfiguraatiota, antaa sen käyttöjärjestelmä palautteen, mikäli konfiguraatiossa on syntaksivirhe tai sen komennossa on käytetty virheellistä argumentin arvoa. Tätä ominaisuutta hyödynnettiin konfiguraation määrittelytiedostosta rakennetun konfiguraation oikeellisuuden tarkistuksessa laitteen kannalta, eli tarkistettiin hyväksyykö laite konfiguraation. Tutkimuksessa käytetyssä laitemallissa oli lisäksi mahdollista hyödyntää erillistä istuntoa, johon aiotun uuden konfiguraation pystyi syöttämään ilman, että se vaikutti ajossa olevaan konfiguraatioon. Laitteen antaman palautteen perusteella epäonnistuneessa hyväksyntätestistä saatiin selville, missä kohtaa konfiguraatiota virhe syntyi.

Verkkoympäristön mallintaminen testiympäristöön konfiguraatiomuutosten osalta vaatii runsaasti joko fyysisiä tai virtuaalisia laiteresursseja. Sitä kannattaakin välttää, kun tehdään muutosta konfiguraationhallinnassa olemassa olevaan palveluun. Nämä muutokset voidaankin tehdä suoraan tuotantoympäristöön, ja hyväksyntätesteillä varmistetaan, että laitteiden palvelukonfiguraatiot toimivat oikein. Sen sijaan kun konfiguraationhallintaan ollaan määrittelemässä uutta palvelua, on testiympäristön luominen kyseisen

kehityksen testaamiseen kannattavaa, koska siihen riittää yleensä muutama testilaite. Tällöin kehitysympäristössä ja käyttöönottoputken kehityshaarassa voidaan toteuttaa uuden palvelun konfiguraatiorakenne, Jinja2-malline sekä palvelun hyväksyntätestit, ennen kuin ne yhdistetään päälinjaan. Tutkimuksessa käytetyssä verkossa tällaisesta kehityksestä ovat esimerkkinä alus- ja kerrosverkon konfiguraatiot ja testit.

5.4 Palautus

5.4.1 Epäonnistunut toimitus tulee pystyä palauttamaan

Ennen konfiguraatiomuutosten tekemistä kohdelaitteille, otetaan niissä sillä hetkellä ajossa olleista konfiguraatioista varmuuskopiot. Jos konfiguraation toimittamisen jälkeen tehty toimivuuden tarkistustestaus epäonnistuu, palautetaan aiemmin käytössä ollut konfiguraatio laitteelle.

5.4.2 Palautuksen toimivuuden testaus säännöllisin väliajoin

Kun laitevalmistajan tekemälle ohjelmistopäivitykselle tehdään hyväksyntätestausta kehitysympäristössä, testaussuunnitelmassa on hyvä olla myös palautuksen testaus. Näin varmistutaan, että ohjelmistopäivitys mahdollistaa edelleen palauttamisen tekemisen nykyisellä skriptillä.

5.5 Konfiguraationhallinta

5.5.1 Yleiskäyttöiset skriptit siten, että samoilla skripteillä toimitetaan sekä testi- että tuotantoympäristöön

Käyttöönottoputki hyödyntää samoja pelikirjoja sekä kehitys- että tuotantoympäristössä, sekä siten myös kehityshaarassa ja päälinjassa. Ympäristön valintaa ohjataan ympäristöjä

varten luoduilla tiedostoilla, joista konfiguraationhallintajärjestelmä kerää pelikirjan suorituksen kohdelaitteet.

5.5.2 Kaksivaiheisella testauksella varmistetaan laitteiden tavoitettavuus ja palveluiden oikeanlainen toiminta.

Aktiviteettisolmuissa, joissa otetaan yhteys kohdelaitteisiin, ensimmäinen vaihe on tarkistaa että yhteys on mahdollista muodostaa. Konfiguraationhallintajärjestelmä ilmoittaa myös itsessään, mikäli se ei saa muodostettua yhteyttä laitteeseen. Erilliset testit ovat kuitenkin nopeampia, jolloin myös aktiviteetti tarvittaessa epäonnistuu nopeasti. Kun konfiguraatiot on toimitettu, käyttöönottoputken lopuksi tehdään hyväksyntätestit, joilla varmistetaan palveluiden oikeanlainen toiminta.

6 Diskussio

Tutkimuskysymyksenä oli: Miten hyödyntää jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia verkkolaitteiden hallintaan lähiverkon palvelutuotannossa? Sitä tutkittiin toteuttamalla toimintamallin teorian pohjalta käyttöönottoputki, joka oli sovitettu tietoverkon laitteiden konfigurointiin huomioimalla alfa-yrityksen toimintaympäristö. Käyttöönottoputki on siten toimintamallin ilmentymä lähiverkon palvelutuotannossa.

Jatkuvaa integrointia ja jatkuvaa toimittamista on käsitelty ja tutkittu sovelluskehityksen näkökulmasta, kuten Fowler (2006), Humble ja Farley (2010) sekä Ståhl ja Bosch (2014a) ovat tehneet. Samalla kulmalla on toimintamalliin tullut mukaan sovelluksen käyttöympäristön konfiguraation hallinta, jota Fowler (2016), Humble ja Farley (2010) sekä Johann (2017) ovat tarkastelleet. Ympäristö koostuu mm. palvelinkäyttöjärjestelmistä, tietokantapalvelimista ja toki myös konesalin tietoliikennelaitteista, mutta lähtökohtana on aina ympäristön saattaminen toimintakuntoon samassa käyttöönottoputkessa toimitettavalle sovellukselle. Pelkästään verkkolaitteiden konfiguraatiolle tehtyä tutkimusta ei kirjallisuuskatsauksen mukaan ole aiemmin tehty. Oman näkökulmansa tutkimukseen tuo ympäristö, jossa alfa-yritys toimii ja johon tutkimusongelma on asetettu. Siinä hallittavia verkkoympäristöjä on useita erilaisia asiakkaasta ja laitemallista riippuen, ja siten palvelutuotantokin on organisoitunut toimimaan prosessimaisesti näiden ympäristöjen ylläpidon osalta.

Kappaleessa 3.3 on listattu Hevnerin ja muiden (2004) muodostamia suosituksia suunnittelutieteellisen tutkimuksen tekemiselle. Käyn seuraavassa läpi suositukset ja miten niitä on noudatettu tutkimuksessa.

Tuloksena artefakti

Tutkimuksen tuloksena syntyi artefakti, joka on mahdollista ottaa käyttöön alfa-yrityksessä sellaisenaan tai soveltaen.

Ongelman olennaisuus

Tutkimus on tehty alfa-yrityksessä tunnistetun ongelman ratkaisemiseksi.

Suunnittelun arviointi

Artefaktia on simuloitu ja sen soveltuvuutta teoriaan perustuviin tavoitteisiin on arvioitu laadullisesti suunnittelusyklin arviointivaiheessa.

Tutkimuksen tieteellinen tuottavuus

Tutkimuksessa rakennettiin ja kuvattiin tietoverkon laitteiden konfigurointiin käyttöönottoputki, joka noudattaa jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia.

Tutkimuksen täsmällisyys

Tutkimus on toteutettu noudattamalla suunnittelutieteellisen tutkimuksen DSRM-prosessimallia. Prosessissa artefaktin suunnittelussa on käytetty Humblen ja Farleyn (2010) viiden kohdan mallia käyttöönottoputken toteuttamiseksi.

Suunnittelu etsintäprosessina

Tutkimuksessa on hyödynnetty jatkuvan integroinnin ja jatkuvan toimittamisen teoriasta asetettuja tavoitteita käyttöönottoputkelle, käytetty konfiguraationhallintajärjestelmää yhdessä käyttöönottoputken kanssa, ja sovitettu käyttöönottoputki alfa-yrityksen toimintaympäristöön.

Tutkimuksen kommunikointi

Tutkimus on esitelty ongelman asettaneelle alfa-yritykselle. Lisäksi tutkimus julkaistaan pro gradu -tutkielmana tutkimusyhteisöjen sekä aihealueen asiantuntijoiden käyttöön.

6.1 Tulokset

Tutkimuksen tuloksena syntynyt käyttöönottoputki on kuvattu kappaleessa 4.4. Seuraavassa käyn läpi tulokset, miten jatkuvan integroinnin ja jatkuvan toimittamisen

sovelluskehityksessä tutkittua teoriaa voidaan hyödyntää verkkolaitteiden konfiguraationhallintaan sovitetussa toimintamallissa.

Palaute testauksesta CI:ssä ja CD:ssä

Nopea palaute jatkuvassa integroinnissa ja jatkuvassa toimittamisessa tapahtuvassa testauksessa on mahdollista toteuttaa käyttöönottoputkessa. Toteutustapa voi riippua käytetystä konfiguraationhallintajärjestelmästä, ja tutkimuksessa käytetyllä Ansiblella oli mahdollista tehdä pelikirjojen testaus tarkistusmoodissa sekä käyttää sille valmistettua koodianalyysityökalua. Nopeat testit tuli kuitenkin jakaa kahteen vaiheeseen, paikallisesti palvelimella suoritettaviin, sekä testeihin jotka suoritetaan ottamalla yhteys konfiguroitavaan laitteeseen.

Nopea kierrosaika ja asiakaspalaute

Muutoksia verkon konfiguraatioon voidaan toimintamallilla toimittaa lyhyellä kierrosajalla, kuten voidaan tehdä manuaalisestikin konfiguroimalla. Tämä on oleellista silloin, kun tietyille muutoksille on sovittu kriittinen toimitusaika esim. samalle päivälle. Asiakaspalaute muutoksista saadaan näissä tapauksissa hyvinkin nopeasti.

Lyhyet kehityssyklit

Mikäli verkon rakenne mahdollistaa, voidaan uusien palveluiden kehittäminen pilkkoa lyhyisiin sykleihin, joissa verkkoon provisioidaan yksi palvelukokonaisuus kerrallaan. Tutkimuksessa näin tehtiin alus- ja kerrosverkon rakentamisen yhteydessä. Yleisesti kehitys voidaan jakaa nopeasti toteutettaviin kokonaisuuksiin, esim. konfiguraation määrittelytiedoston rakenne, Jinja2-mallineiden määrittely sekä testien määrittely. Koodin piilottamista on mahdollista hyödyntää siten, että keskeneräistä konfiguraatiomäärittelyä ei provisioida tuotantoverkkoon, ennen kuin palvelun osalta olennaiset kokonaisuudet ovat valmiit.

Työnkulun välittömyys ja läpinäkyvyys

Verkonhallintaympäristössä käyttöönottoputki on oltava kaikkien palvelutuotantoon osallistuvien tiimien henkilöiden käytettävissä. Tutkimuksessa käytetyllä jatkuvan integroinnin ja konfiguraationhallinnan työkaluilla tämä tavoite oli helposti toteutettavissa. Kun palveluhallintaa tehdään lisäksi useille asiakkaille ja erilaisiin verkkoympäristöihin, käyttöönottoputki on hyvä olla mahdollisimman yleisesti soveltuva eri ympäristöjen kesken. Tutkimuksen käyttöönottoputki on toteutettu niin, että muuttuvat tiedot sisältyvät ainoastaan konfiguraationhallinnassa määriteltyyn rakenteeseen.

Päälinjan käyttö ja eheys

Versionhallinnassa käytetään kehityshaaroja sekä tuotantoympäristön konfiguraatiomuutoksiin että uusien palveluiden kehitykseen. Näin varmistetaan, että päälinja pysyy toimintakuntoisena ja mahdollistaa muutosten toimittamisen nopeallakin aikataululla. Kehityshaarassa suoritetut integrointitestit helpottavat tiimien työskentelyä eri asiakasympäristöissä, kun jäsenet voivat olla varmoja, että vaaditut testit suoritetaan ennen yhdistämistä päälinjaan. Kehitysympäristössä tällä metodilla voidaan lisäksi tehdä kaikki testit päälinjasta erillään. Päälinja on siten pyritty pitämään mahdollisimman eheänä. Sekä tuotanto- että kehitysympäristöön konfiguraatiot toimitetaan samoilla pelikirjoilla.

Suoritusaika

Tutkimuksen testiympäristössä koko käyttöönottoputken suoritusaika oli alle kahdeksan minuuttia kun muutos tehtiin tuotantoympäristöön. Tämä alittaa sovelluskehityksessä annetun ohjenuoran kymmenen minuutin suoritusajasta integrointitesteille. Kummassakin tapauksessa muutoksen tekijä saa ilmoitetussa ajassa varmuuden muutoksen onnistumisesta.

Yksikkö-, komponentti- ja hyväksyntätestit, palautuksen toimivuuden testaus

Testitapausten osalta verkonhallinnassa tulee testausta lähestyä hieman eri tavalla. Yksikkötestien sijaan voidaan hyödyntää konfiguraatiohallintajärjestelmän mahdollisuuksia, kuten esim. Ansiblella voidaan pelikirjojen testaus tehdä tarkistusajona. Lisäksi voitiin

käyttää ansible-lint -ohjelmaa tarkistamaan syntaksi. Komponenttitestauksessa voidaan laitteella käytössä olevan käyttöjärjestelmän versiota ajatella komponenttina. Sen päivityksessä tulisikin ensin suorittaa testaus kehitysympäristössä, jotta konfiguroidut palvelut varmasti toimivat. Tässä on hyvä testata myös, että konfiguraation palautus toimii, kuten se on käyttöönottoputkeen määritelty. Mikäli ympäristössä käytetty laite mahdollistaa konfiguraation syöttämisen ennen sen käyttöönottoa, voidaan tätä ominaisuutta hyödyntää hyväksyntätestauksena verkkolaitteen näkökulmasta. Laite antaa palautteen, mikäli konfiguraation syntaksi tai sen argumenttien arvot ovat virheellisiä.

Konfiguraation palautus

Konfiguraation palautuksessa hyödynnettiin automaattisia toimitetun konfiguraation tarkistustestejä. Laitteen käytössä ollut konfiguraatio kopioitiin GitLab Runner -palvelimelle, ennen konfiguraation päivittämistä. Mikäli jokin toimituksen tarkistustesti epäonnistui, voitiin konfiguraatio palauttaa takaisin laitteelle.

6.2 Johtopäätökset

Jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallia hyödyntämällä on mahdollista toteuttaa käyttöönottoputki, jolla verkkolaitteiden konfiguraatioita voidaan kehittää, testata, provisioida ja todentaa toimiviksi. Verrattuna manuaalisesti tehtävään provisointiin, konfiguraatioiden ajaminen verkkolaitteisiin tapahtuu toimintamallin avulla täsmällisesti sekä useisiin laitteisiin yhtäaikaisesti skaalautuen. Mikäli konfiguraatiotiedoissa on virheitä, muutoksen tekijä saa nopean palautteen heti käyttöönottoputken alussa ja hitaammista testeistä sen lopussa, kuten Humble ja Farley (2010) sekä Ståhl ja Bosch (2014a) ovat määritelleet. Lisäksi toimintamallin avulla on mahdollista hyödyntää määrämuotoista nykyisen ja korvaavan konfiguraation vertailua ennen muutoksen käyttöönottoa, mikä muutoin tapahtuisi henkilöstä riippuvalla tarkistusmenetelmällä.

Provisioinnin jälkeinen automaattinen, konfiguroitujen palveluiden toiminnallisuuden testaus on oleellinen osa toimintamallia verkkolaitteiden konfiguraatiossa. Se on

huomattava parannus manuaalisesti tehtäville ja silmämääräisesti tarkistettaville testeille. Verkkolaitteiden konfiguraatiossa muutoksen tekijä saa varmuuden muutoksen onnistumisesta vasta, kun koko käyttöönottoputki on suoritettu. Tämä poikkeaa ohjelmistokehityksestä, jossa varmuuden saamiseen riittää integraatiovaiheen testien läpäisy (Fowler, 2006; Humble &Farley, 2010). Kattava testaus tuo tekijälle varmuuden muutoksen onnistumisesta, mikä on Humblen ja Farleyn (2010) mukaan yksi jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin oleellisista hyödyistä. Toisaalta testit on pystyttävä suunnittelemaan niin, että niiden määrittelystä ei aiheudu liikaa ylimääräistä työtä, kuten alfa-yritykselle tehdyn simulaation yhteydessä huomautettiin. Jos ne tehdään liian tarkoiksi, voi niitä joutua muokkaamaan aina, kun konfiguraatioon tehdään muutoksia.

Toimintamallin hyödyntäminen siten, että päästäisiin nopeaan asiakaspalautteeseen, kuten Chen (2017), Leppänen ja muut (2015) sekä Ohlsson ja muut (2012) ovat todenneet, tarkoittaisi sitä, että verkko olisi jatkuvassa muutostilassa. Tällöin verkosta ei tarvitse päivittää tiettyä teknologiaa kerrallaan, vaan muutokset voidaan tehdä esim. vikaantuneiden laitteiden korvaamisen yhteydessä tai verkon tietyissä osissa tarvittavien uusia palveluja tukevien laitteiden osalta. Jotta tämä olisi mahdollista vähintään riittävien muutoksen jälkeisten testien osalta, tulisi myös vanha toteutus kuvata konfiguraationhallintaan, mikäli sitä ei ole palveluntuottajalla kuvattuna ennestään.

6.3 Rajoitukset

Konfiguraationhallintajärjestelmän käyttö verkkolaitteiden konfiguraatioon soveltuu hyvin yhteen jatkuvan integroinnin ja jatkuvan toimittamisen toimintamallin kanssa. Tutkimuksessa hallintajärjestelmään tehtiin yhdenlainen konfiguraatioiden määrittelyiden rakenne, jolla oli mahdollista konfiguroida testiympäristöä. Rakenne ei kuitenkaan ole optimaalinen monen asiakkaan, sekä eri laitemallien ja -tekniikoiden ympäristössä. Millainen rakenne on helposti hallittava ja silti toimiva tällaisessa ympäristössä, voisi olla oma tutkimuksen kohteensa.

Lähteet

Ansible. (2016, 28. tammikuuta). Use Case – Configuration Management. Ansible. Noudettu 2020-02-15 osoitteesta https://www.ansible.com/use-cases/configuration-management

Ansible. (2020a, 3. huhtikuuta). Ansible for Network Automation. Ansible. Noudettu 2020-04-11 osoitteesta https://docs.ansible.com/ansible/latest/network/index.html

Ansible. (2020b, 17 huhtikuuta). Templating (Jinja2). Ansible. Noudettu 2020-04-18 osoitteesta https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html

Ansible. (2020c). Ansible Lint Documentation. Read the Docs. Noudettu 2020-04-18 osoitteesta https://ansible-lint.readthedocs.io/en/latest/

Arista. (2017). vEOS-lab Data Sheet. Arista. Noudettu 2020-03-19 osoitteesta https://www.arista.com/assets/data/pdf/Datasheets/vEOS_Datasheet.pdf

Chacon, S., & Straub, B. (2020, 13. huhtikuuta). Pro Git. Noudettu 2020-04-15 osoitteesta https://git-scm.com/book/en/v2 (Alkuperäinen teos julkaistu 2014).

Chen, L. (2017). Continuous Delivery: Overcoming adoption challenges. *The Journal of Systems and Software*, 128, 72–86. https://doi.org/10.1016/j.jss.2017.02.013

Cisco Systems. (2018, 19. marraskuuta). Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper. Cisco Systems. Noudettu 2020-02-15 osoitteesta https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

Cisco Systems. (2019a, 27. helmikuuta). Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. Cisco Systems. Noudettu 2020-02-15 osoitteesta https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

Cisco Systems. (2019b). 2020 Global Networking Trends Report. Cisco Systems. Noudettu 2019-12-15 osoitteesta https://www.cisco.com/c/dam/m/en_us/solutions/enterprise-networks/networking-report/files/GLBL-ENG_NB-06_0_NA_RPT_PDF_MOFU-no-NetworkingTrendsReport-NB_rpten018612_5.pdf?ccid=cc001244&oid=rpten018612

Clark, S. (2018, 13. helmikuuta). Continuous Integration and Deployment for the Network. Cisco Systems. Noudettu 2020-02-22 osoitteesta https://blogs.cisco.com/developer/continuous-integration-and-deployment-for-the-network

Coleman, A., Bombal, D., Duponchelle, J. (2019, 24. joulukuuta). Getting Started with GNS3. GNS3. Noudettu 2020-03-19 osoitteesta https://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html

Fowler, M. (2006, 1. toukokuuta). Continuous Integration. Noudettu 2020-01-17 osoitteesta https://martinfowler.com/articles/continuousIntegration.html

Fowler, M. (2012, 10. heinäkuuta). SnowflakeServer. Noudettu 2020-03-17 osoitteesta https://martinfowler.com/bliki/SnowflakeServer.html

Fowler, M. (2016, 1. marraskuuta). InfrastructureAsCode. Noudettu 2020-03-17 osoitteesta https://martinfowler.com/bliki/InfrastructureAsCode.html

GitLab. (2019, 30. huhtikuuta). GitLab CI/CD. GitLab Docs. Noudettu 2020-03-16 osoitteesta https://docs.gitlab.com/12.9/ee/ci/README.html

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19(2), 1–6.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *Mis Quarterly*, 28(1), 75–105.

Humble, J., Farley, D. (2010). Continuous Delivery: Reliable Software Releases Through Build, Test And Deployment Automation. Addison-Wesley.

Johann, S. (2017). Kief Morris on Infrastructure as Code. *IEEE Software*. 34(1), 117–120. https://doi.org/10.1109/MS.2017.13

Juniper Networks. (2019). 2019 State of Network Automation Report. Juniper Networks. Noudettu 2020-02-15 osoitteesta https://www.juniper.net/us/en/forms/sonar/

Kelly, R. (2018, 6. elokuuta). A Detailed Overview of the EVPN & VxLAN Protocols, Route Types, Use-Cases and Architectures. Arista. Rajattu pääsy. Noudettu 2020-03-23 osoitteesta https://eos.arista.com/evpn-vxlan-design-guide

Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V-P., Itkonen, J., Mäntylä, M. V., & Männistö, T. (2015). The Highways and Country Roads to Continuous Deployment. *IEEE Software*. 32(2), 64–72. https://doi.org/10.1109/MS.2015.50

March, S. T., Smith, G. F. (1995). Design and Natural Science Research on Information Technology. *Decision Support Systems*, 15(4), 251–266.

Olsson, H. H., Alahyari, H., & Bosch, J. (2012). Climbing the “Stairway to Heaven” – A multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. *38th Euromicro Conference on Software Engineering and Advanced Applications*, 392–399. https://doi.org/10.1109/SEAA.2012.54

Oren, B-K., Evans, C. & döt Net, I. (2009, 1. lokakuuta). YAML Ain’t Markup Language (YAML™) Version 1.2. Yaml.org. Noudettu 2020-03-16 osoitteesta https://yaml.org/spec/1.2/spec.html

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), 45–77.

Puppet. (2019, 23. marraskuuta). Configuration Management. Puppet. Noudettu 2020-02-15 osoitteesta https://puppet.com/use-cases/configuration-management/

Ståhl, D., & Bosch, J. (2014a). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 87, 48–59. https://doi.org/10.1016/j.jss.2013.08.032

Ståhl, D., & Bosch, J. (2014b). Automated Software Integration Flows in Industry: A Multiple-Case Study. *Publication:* ICSE Companion 2014: Companion Proceedings of the 36th International Conference on Software Engineering, 54–63. https://doi.org/10.1145/2591062.2591186

Liitteet

Liite 1. Käyttöönottoputken .gitlab-ci.yml -konfiguraatio

```
stages:
  - kommitointitestit
  - rakentaminen
  - ssh-avaimen skannaus
  - toimittamisen tarkistus
  - hyväksyntätestaus
  - toimittaminen

# Tallenna Python virtuaaliympäristö välimuistiin.
cache:
  paths:
    - venv/

# Aseta vakioidut Python ja Ansible -ympäristöt jokaisen
# aktiviteetin yhteydessä GitLab Runner -palvelimelle.
before_script:
  - python3 -m venv venv
  - source venv/bin/activate
  - python --version  # Tulosta versio virheenkorjaukseen.
  - pip install -r requirements.txt
  - ansible --version  # Tulosta versio virheenkorjaukseen.

# Testaa rakentamisen pelikirja.
Rakentamisen tarkistus:
  stage: kommitointitestit
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml rakenna_konfiguraa-
tio.yml --limit @_tuotanto_kohdelaitteet --vault-password-
file ~/.vault_pass.txt --check --diff
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_tuotanto_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_tuotanto_kohdelaitteet
  tags:
    - tuotanto

# Testaa rakentamisen pelikirja.
kehitys/Rakentamisen tarkistus:
  stage: kommitointitestit
  script:
    - cd ansible
```

```
    - ansible-playbook -i inventory.yml rakenna_konfiguraa-
tio.yml --limit @_kehitys_kohdelaitteet --vault-password-
file ~/.vault_pass.txt --check --diff
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_kehitys_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_kehitys_kohdelaitteet
  tags:
    - kehitys

# Pelikirjojen syntaksi- ja moduulitarkistus
Koodin laadun tarkistus:
  stage: kommitointitestit
  script:
    - cd ansible
    - ansible-lint *.y*ml group_vars/*.y*ml host_vars/*.y*ml
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_tuotanto_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_tuotanto_kohdelaitteet
  tags:
    - tuotanto

# Pelikirjojen syntaksi- ja moduulitarkistus
kehitys/Koodin laadun tarkistus:
  stage: kommitointitestit
  script:
    - cd ansible
    - ansible-lint *.y*ml group_vars/*.y*ml host_vars/*.y*ml
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_kehitys_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_kehitys_kohdelaitteet
  tags:
    - kehitys

# Rakenna konfiguraatiotiedostot ja tallenna ne artefakteiksi.
Rakentaminen:
  stage: rakentaminen
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml rakenna_konfiguraa-
tio.yml --limit @_tuotanto_kohdelaitteet --vault-password-
file ~/.vault_pass.txt
```

```
  artifacts:
    paths:
      - ansible/konfiguraatiot/
    expire_in: 1 week
  needs:
    - Rakentamisen tarkistus
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_tuotanto_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_tuotanto_kohdelaitteet
  tags:
    - tuotanto

# Rakenna konfiguraatiotiedostot ja tallenna ne artefakteiksi.
kehitys/Rakentaminen:
  stage: rakentaminen
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml rakenna_konfiguraa-
tio.yml  --limit  @_kehitys_kohdelaitteet  --vault-password-
file ~/.vault_pass.txt
  artifacts:
    paths:
      - ansible/konfiguraatiot/
    expire_in: 1 week
  needs:
    - kehitys/Rakentamisen tarkistus
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_kehitys_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_kehitys_kohdelaitteet
  tags:
    - kehitys

# SSH-avaimen skannaus
SSH-avaimen skannaus:
  stage: ssh-avaimen skannaus
  script:
    - cd ansible
    -  ansible-playbook  -i  inventory.yml  ssh_avaimen_skan-
naus.yml --limit @_ssh_avaimen_skannaus_tuotantolaitteet --
vault-password-file ~/.vault_pass.txt
  rules:
    - if: $CI_MERGE_REQUEST_ID
      when: never
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
```

```
        - ansible/_ssh_avaimen_skannaus_tuotantolaitteet
  tags:
    - tuotanto

# SSH-avaimen skannaus
kehitys/SSH-avaimen skannaus:
  stage: ssh-avaimen skannaus
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml ssh_avaimen_skan-
naus.yml --limit @_ssh_avaimen_skannaus_kehityslaitteet --
vault-password-file ~/.vault_pass.txt
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_ssh_avaimen_skannaus_kehityslaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_ssh_avaimen_skannaus_kehityslaitteet
  tags:
    - kehitys

# Testaa toimittamisen pelikirja.
Toimittamisen tarkistus:
  stage: toimittamisen tarkistus
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml tarkista_yhteys.yml
--limit @_tuotanto_kohdelaitteet --vault-password-file
~/.vault_pass.txt
    - ansible-playbook -i inventory.yml toimita_konfiguraa-
tio.yml --limit @_tuotanto_kohdelaitteet --vault-password-
file ~/.vault_pass.txt --check --diff
  dependencies:
    - Rakentaminen
  rules:
    - if: $CI_MERGE_REQUEST_ID
      when: never
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_tuotanto_kohdelaitteet
  tags:
    - tuotanto

# Testaa toimittamisen pelikirja.
kehitys/Toimittamisen tarkistus:
  stage: toimittamisen tarkistus
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml tarkista_yhteys.yml
--limit @_kehitys_kohdelaitteet --vault-password-file
~/.vault_pass.txt
```

```
    - ansible-playbook -i inventory.yml toimita_konfiguraa-
tio.yml  --limit  @_kehitys_kohdelaitteet  --vault-password-
file ~/.vault_pass.txt --check --diff
  dependencies:
    - kehitys/Rakentaminen
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_kehitys_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_kehitys_kohdelaitteet
  tags:
    - kehitys

# Testaa, että laite hyväksyy konfiguraation.
Aiotun konfiguraation hyväksyntä:
  stage: hyväksyntätestaus
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml tarkista_yhteys.yml
--limit    @_tuotanto_kohdelaitteet    --vault-password-file
~/.vault_pass.txt
    - ansible-playbook -i inventory.yml testaa_aiottu_konfi-
guraatio.yml --limit @_tuotanto_kohdelaitteet --vault-pass-
word-file ~/.vault_pass.txt
  dependencies:
    - Rakentaminen
  rules:
    - if: $CI_MERGE_REQUEST_ID
      when: never
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_tuotanto_kohdelaitteet
  tags:
    - tuotanto

# Testaa, että laite hyväksyy konfiguraation.
kehitys/Aiotun konfiguraation hyväksyntä:
  stage: hyväksyntätestaus
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml tarkista_yhteys.yml
--limit    @_kehitys_kohdelaitteet    --vault-password-file
~/.vault_pass.txt
    - ansible-playbook -i inventory.yml testaa_aiottu_konfi-
guraatio.yml  --limit  @_kehitys_kohdelaitteet  --vault-pass-
word-file ~/.vault_pass.txt
  dependencies:
    - kehitys/Rakentaminen
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
```

```
        - ansible/_kehitys_kohdelaitteet
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_kehitys_kohdelaitteet
  tags:
    - kehitys

# Toimita konfiguraatio laitteille ja tarkista konfiguraation
# toimivuus.
Konfiguraation toimittaminen:
  stage: toimittaminen
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml tarkista_yhteys.yml
--limit    @_tuotanto_kohdelaitteet    --vault-password-file
~/.vault_pass.txt
    - ansible-playbook -i inventory.yml varmuuskopioi_konfi-
guraatio.yml --limit @_tuotanto_kohdelaitteet --vault-pass-
word-file ~/.vault_pass.txt
    - ansible-playbook -i inventory.yml toimita_konfiguraa-
tio.yml  --limit  @_tuotanto_kohdelaitteet  --vault-password-
file ~/.vault_pass.txt
    - ansible-playbook -i inventory.yml toimituksen_tarkis-
tus.yml  --limit  @_tuotanto_kohdelaitteet  --vault-password-
file ~/.vault_pass.txt
  dependencies:
    - Rakentaminen
  rules:
    - if: $CI_MERGE_REQUEST_ID
      when: never
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_tuotanto_kohdelaitteet
      when: manual
  tags:
    - tuotanto


# Toimita konfiguraatio laitteille ja tarkista konfiguraation
# toimivuus.
kehitys/Konfiguraation toimittaminen:
  stage: toimittaminen
  script:
    - cd ansible
    - ansible-playbook -i inventory.yml tarkista_yhteys.yml
--limit    @_kehitys_kohdelaitteet    --vault-password-file
~/.vault_pass.txt
    - ansible-playbook -i inventory.yml varmuuskopioi_konfi-
guraatio.yml  --limit  @_kehitys_kohdelaitteet  --vault-pass-
word-file ~/.vault_pass.txt
    - ansible-playbook -i inventory.yml toimita_konfiguraa-
tio.yml  --limit  @_kehitys_kohdelaitteet  --vault-password-
file ~/.vault_pass.txt
```

```
    - ansible-playbook -i inventory.yml toimituksen_tarkis-
tus.yml  --limit  @_kehitys_kohdelaitteet  --vault-password-
file ~/.vault_pass.txt
  dependencies:
    - kehitys/Rakentaminen
  rules:
    - if: $CI_MERGE_REQUEST_ID
      exists:
        - ansible/_kehitys_kohdelaitteet
      when: manual
    - if: $CI_COMMIT_BRANCH == 'master'
      exists:
        - ansible/_kehitys_kohdelaitteet
      when: manual
  tags:
    - kehitys
```